# CS 2211 Systems Programming

Part Ten: Structures

		397 - 398 - 399 -	
		101110011011100110001101	l
		400 - 401 - 402 -	
		00100011011001110011111111	1
		403 - 404 - 405 -	
		010101111010101101000110	)
		406 - 407 - 408 -	
Address Label	Label	11100110111001100011001	)
		409 - 410 - 411 -	
		0010001101100111001111111111	l
		412 - 413 - 414 -	
		101110011011100110001101	1
		415 - 416 - 417 -	
		11100110111001100011001	)
		418 - 419 - 420 -	
		01010111101010110101000110	)
		421 - 422 - 423 -	
		10111001101110011000110	1

# 结构体中允许给着存不过类型毒品指

## **MEMORY MAPPING - STRUCTURE**

- a construct to group together dissimilar variables under one name 1011100111011100110001101 杨莹(tag).  $0\;0\;1\;0\;0\;0\;1\;1\;0\;1\;1\;0\;0\;1\;1\;0\;0\;1\;1\;0\;1\;1\;1$ 01010111101010110101000110 111001101110011000110010 /\* "person" is name for structure type \*/ 1 1 struct person { 0 1 member { char first[32]; /\* 1st field is array of char \*/ char last[32]; /\* 2nd field is array of char \*/ int year; /\* 3rd field is int \*/ double ppg; /\* 4th field is double \*/ 1 0 1 0 0 1 /\* ending ; \*/ /\* means end of structure type definition \*/ variable

- a construct to **group** together

dissimilar variables under one name



Address Label Labe

#### NOTE:

This code does NOT create a variable (similar to Class declaration in OOP)

- grandfather of this concept
- this is ONLY a template for a variable declaration.
- this creates a **USER DEFINED** variable typ

- a construct to **group** together

dissimilar variables under one name

```
101110011011100110001101
/* "person" is name for structure type */
                                                         1 1
struct person {
                                                         1 0
       char first[32]; /* 1st field is array of char */
       char last[32]; /* 2nd field is array of char */
                                                         1 1
       int year; /* 3rd field is int */
      double ppg; /* 4th field is double */
                                                         0 1
             /* ending ; */
       };
                                                         1 0
/* means end of structure type definition */
                                                         1 0
                                                         0 1
int i;
struct person teacher;
```

creates an integer variable (space to store an integer) i - and creates a person variable (space to store two character arrays, and integer and a double) teacher.

```
/* "person" is name for structure type */
struct person {
    char first[32]; /* 1st field is array of char */
    char last[32]; /* 2nd field is array of char */
    int year; /* 3rd field is int */
    double ppg; /* 4th field is double */
    }; /* means end of structure type definition */
```

Address Label	Label	Address	Value

struct person teacher;

- memory 400 to 475

the struct person takes up (requires)76 bytes of data

Address Lab	el	Label	Address	Value
teacher.	first	teacher	400 -	
_		teacher.first[0]		
		teacher.first[1]	401 -	
		teacher.first[2]	402 -	
		teacher.first[3]	403 -	
		teacher.first[5]	404 -	
		teacher.first[6] – [31]	405 - 431	
teacher	last	teacher.last[0]	432 -	
		teacher.last[1]	433 -	
	1	teacher.last[2]	434 -	
		teacher.last[3]	435 -	
		teacher.last[4]	436 -	
t		teacher.last[5]	437 -	
		teacher last[5]	/138 -	

#### note:

teacher
teacher.first
&teacher.first[0]

all returns the address 400

teacher.last[2]	434 -	
teacher.last[3]	435 -	
teacher.last[4]	436 -	
teacher.last[5]	437 -	
teacher.last[5]	438 -	
teacher.last[6] – [31]	439 - 463	
teacher.year	464 - 467	
teacher.ppg	468 - 475	

struct person teacher;

teacher.year=2005;
teacher.ppg=10.4;
strcpy(teacher.first,"Sam");
strcpy(teacher.last,"Maggs");

teacher.first       400 -         teacher.first[0]       s         teacher.first[1]       401 -         teacher.first[2]       402 -         teacher.first[3]       403 -         teacher.first[5]       404 -         teacher.first[6] - [31]       405 - 431         teacher.last[0]       432 -         M       433 -         teacher.last[1]       433 -         teacher.last[2]       434 -         g       435 -         teacher.last[3]       435 -         teacher.last[4]       436 -         teacher.last[5]       437 -         teacher.last[5]       438 -	Address Label	Label	Address	Value
teacher.first[1]	teacher.first	teacher	400 -	
teacher.first[2] 402 - m  teacher.first[3] 403 - \0  teacher.first[5] 404 -  teacher.first[6] - [31] 405 - 431  teacher.last teacher.last[0] 432 - M  teacher.last[1] 433 - a  teacher.last[2] 434 - g  teacher.last[3] 435 - g  teacher.last[4] 436 - s  teacher.last[5] 437 - \0  teacher.last[5] 438 -		teacher.first[0]		S
teacher.first[3] 403 - \ \ \teacher.first[5] 404 - \ \ \teacher.first[6] - [31] 405 - 431 \\ \teacher.last teacher.last[0] 432 - \ \ \teacher.last[1] 433 - \ \teacher.last[2] 434 - \ \ \teacher.last[3] 435 - \ \ \teacher.last[4] 436 - \ \teacher.last[5] 437 - \ \ \teacher.last[5] 438 - \\ \teacher.last[		teacher.first[1]	401 -	a
teacher.first[5] 404 - teacher.first[6] - [31] 405 - 431  teacher.last teacher.last[0] 432 - teacher.last[1] 433 - teacher.last[2] 434 - teacher.last[3] 435 - teacher.last[4] 436 - teacher.last[5] 437 - teacher.last[5] 438 -		teacher.first[2]	402 -	m
teacher.first[6] - [31]		teacher.first[3]	403 -	\0
teacher.last       teacher.last[0]       432       -       M         teacher.last[1]       433       -       a         teacher.last[2]       434       -       g         teacher.last[3]       435       -       g         teacher.last[4]       436       -       s         teacher.last[5]       437       -       \0         teacher.last[5]       438       -		teacher.first[5]	404 -	
teacher.last[1] 433 - a teacher.last[2] 434 - g teacher.last[3] 435 - g teacher.last[4] 436 - a teacher.last[5] 437 - \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \		teacher.first[6] – [31]	405 - 431	
teacher.last[2] 434 - g teacher.last[3] 435 - g teacher.last[4] 436 - s teacher.last[5] 437 - \0 teacher.last[5] 438 -	teacher.last	teacher.last[0]	432 -	M
teacher.last[3] 435 - g  teacher.last[4] 436 - s  teacher.last[5] 437 - \0  teacher.last[5] 438 -		teacher.last[1]	433 -	a
teacher.last[4] 436 - 5 teacher.last[5] 437 - 0 teacher.last[5] 438 -		teacher.last[2]	434 -	g
teacher.last[5] 437 - \0 teacher.last[5] 438 -		teacher.last[3]	435 -	g
teacher.last[5] 438 -		teacher.last[4]	436 -	S
		teacher.last[5]	437 -	\0
toachor last[6] _ [21]		teacher.last[5]	438 -	
teacher.iast[0] = [31] 433 - 403		teacher.last[6] – [31]	439 - 463	
teacher.year 464 - 467 2005		teacher.year	464 - 467	2005
teacher.ppg 468 - 475 <b>10 . 4</b>		teacher.ppg	468 - 475	10.4

/\* "norson" is name for structure type \*/

## notice memory location 400

```
teacher // refers to the entire structure starting at 400 teacher.first // refers to the character array starting at 400 teacher.first[0] // refers to the single character byte at 400
```

Having a label that refers to the entire structure is used in two ways:

1.) allows assignment between two identical structure types:

```
struct person mailman, teacher;
mailman = teacher;
    // a byte by bytes transference of values (including nulls)
```

2.) passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

teacher.last[5]	438 -	
teacher.last[6] – [31]	439 - 463	
teacher.year	464 - 467	2005
teacher.ppg	468 - 475	10.4

- passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

```
int
douk
};
/* means end of
struct person

teacher.year=200
teacher.ppg=10.4
strcpy(teacher.f
strcpy(teacher.f)
```

```
douk DisplayStats (struct person Input) {
          Input.first = "Dunsul";
   /* "person" is name for structure type */
   struct person {
                first[32]; /* 1st field is array of char */
           char
           char last[32]; /* 2nd field is array of char */
           int year; /* 3rd field is int */
           double ppg;
                                /* 4th field is double */
                                 /* ending ; */
           } ;
   /* means end of structure type definition */
   struct person teacher;
   teacher.year=2005;
   teacher.ppg=10.4;
   strcpy(teacher.first, "Sam");
   strcpy(teacher.last, "Maggs");
    DisplayStats (teacher);
    printf("%s\n", teacher.first);
```

#### **SIDE BAR:**

is the a callby valueor-a call byreference?

- passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

```
int douk pisp // Disp // Disp
```

```
DisplayStats(struct person Input) {
       Input.first = "Dunsul";
/* "person" is name for structure type */
struct person {
                first[32]; /* 1st field is array of char */
        char
        char last[32]; /* 2nd field is array of char */
                           /* 3rd field is int */
        int
              year;
        double ppg;
                            /* 4th field is double */
                             /* ending ; */
        };
/* means end of structure type definition */
struct person teacher;
```

### **SIDE BAR:**

is the a callby valueor-a call byreference ?

teacher.year=2005;
teacher.ppg=10.4;
<pre>strcpy(teacher.first,"Sam");</pre>
strcpy(teacher.last,"Maggs");
DisplayStats(teacher);

printf("%s\n", teacher.first);

Label	Address	Value

- passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

```
int douk
};

/* means end of

struct person

teacher.year=200
teacher.ppg=10.4
strcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.istrcpy(teacher.
```

```
DisplayStats(struct person Input) {
    Input.first = "Dunsul";
}

con

-220

/* "person" is name for structure type */
struct person {
    char first[32]; /* 1st field is array of char */
    char last[32]; /* 2nd field is array of char */
    int year; /* 3rd field is int */
    double ppg; /* 4th field is double */
    }; /* ending; */

/* means end of structure type definition */
```

### **SIDE BAR:**

is the a callby valueor-a call byreference ?

#### struct person teacher;

teacher.year=2005;
teacher.ppg=10.4;
<pre>strcpy(teacher.first,"Sam");</pre>
<pre>strcpy(teacher.last,"Maggs");</pre>
DisplavStats(teacher);

printf("%s\n", teacher.first);

Label	Address	Value
teacher	400 - 475	

teacher.ppg

468 - 475

10.4

- passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

```
/* means end of
struct person
teacher.vear=200
teacher.ppg=10.4
strcpv(teacher.:
strcpy(teacher.]
```

```
DisplayStats(struct person Input) {
       Input.first = "Dunsul";
/* "person" is name for structure type */
struct person {
             first[32]; /* 1st field is array of char */
        char
        char last[32]; /* 2nd field is array of char */
        int year; /* 3rd field is int */
        double ppg;
                            /* 4th field is double */
                             /* ending ; */
        } ;
/* means end of structure type definition */
struct person teacher;
```

Label

### SIDE BAR:

- is the a call by value -ora call by reference?

teacher.year=2005;
teacher.ppg=10.4;
<pre>strcpy(teacher.first,"Sam");</pre>
strcpy(teacher.last,"Maggs");

printf("%s\n", teacher.first);

DisplayStats(teacher);

teacher	400 - 475	[original]

**Address** 

Value

- passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

intr
douk
};
/* means end of
struct person
teacher.year=200
teacher.ppg=10.4
strcpy(teacher.:
strcpy(teacher.l

17027.	/* 3rd fiold	ic int */		
DisplayStats(st	ruct person	Input) {		
Input.fi	rst = "Dunsu	ıl";		
1				

	}	Address Label	Label	Address	Value
1	/ h •••	teacher.first	teacher	400 -	
200			teacher.first[0]		S
) . 4 : . 1	struct per ch		teacher.first[1]	401 -	a
:.:	ch		teacher.first[2]	402 -	III.
	ir		teacher.first[3]	403 - (	( <del>)</del>
	dc		teacher.first[5]	404 -	
	} ;		teacher.first[6] – [31]	405 - 431	
	/* means e	teacher.last	teacher.last[0]	432 -	M
	at might in a		teacher.last[1]	433 -	a
	struct pe		teacher.last[2]	434 -	9
	teacher.ye		teacher.last[3]	435 -	g
	teacher.pp		teacher.last[4]	436 -	S
	strcpy(tea		teacher.last[5]	437 -	\0
	strcpy(tea		teacher.last[5]	438 -	
			teacher.last[6] – [31]	439 - 463	
<sup>1</sup>	DisplaySt		teacher.year	464 - 467	2005
	printf("%		teacher.ppg	468 - 475	10.4

## **SIDE BAR:**

is the a callby valueor-a call byreference ?

- passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

```
/* means end of
struct person
teacher.vear=200
teacher.ppg=10.4
strcpy(teacher.f
strcpy(teacher.]
```

```
douk DisplayStats(struct person Input) {
         Input.first = "Dunsul";
   /* "person" is name for structure type */
   struct person {
           char first[32]; /* 1st field is array of char */
           char last[32]; /* 2nd field is array of char */
           int year; /* 3rd field is int */
           double ppg;
                              /* 4th field is double */
                               /* ending ; */
           };
   /* means end of structure type definition */
   struct person teacher;
```

## SIDE BAR:

- is the a call by value -ora call by reference?

teacher.year=2005;
teacher.ppg=10.4;
<pre>strcpy(teacher.first,"Sam");</pre>
<pre>strcpy(teacher.last,"Maggs");</pre>

teacher	400 - 475	[original]
Input	710 - 785	[copy]
		<b>A</b>

**Address** 

DisplayStats(teacher); printf("%s\n", teacher.first);

Value

teacher.ppg

Label

468 - 475

- passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

```
int
douk
};

/* means end of

struct person

teacher.year=200
teacher.ppg=10.4
strcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teacher.isstrcpy(teach
```

```
DisplayStats(struct person Input) {
    Input.first = "Dunsul";
    }

/* "person" is name for structure type */
struct person {
        char first[32]; /* 1st field is array of char */
        char last[32]; /* 2nd field is array of char */
        int year; /* 3rd field is int */
        double ppg;
    };

/* means end of struc

struct person teacher;
```

## **SIDE BAR:**

is the a callby valueor-a call byreference ?

teacher.year=2005;
teacher.ppg=10.4;
<pre>strcpy(teacher.first,"Sam");</pre>
<pre>strcpy(teacher.last,"Maggs");</pre>

Labei	Address	value
teacher	400 - 475	[original]
Input	710 - 785	[copy]

DisplayStats(teacher);
printf("%s\n", teacher.first);

- passing a structure as a parameter to a function.

// allows the easy transfer of large amounts of data

/\* means end of structure type definition \*/

```
int
    doul
    /* means end of
    struct person
    teacher.year=20(
    teacher.ppg=10.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.strcpy(teacher.
```

DisplayStats (teacher);

printf("%s\n", teacher.first);

## **SIDE BAR:**

is the a callby valueor-a call byreference?

struct person teacher;			
	Label	Address	Value
teacher.year=2005;			
teacher.ppg=10.4;	teacher	400 - 475	[original]
<pre>strcpy(teacher.first,"Sam");</pre>	Input	710 - 785	[copy]
<pre>strcpy(teacher.last,"Maggs");</pre>			

teacher.ppg

468 - 475

1st field is array of char \*/

2nd field is array of char \*/

/\* 3rd field is int \*/

 $^{\prime *}$  4th field is double  $^{\star \prime}$ 

10.4

teacher = Display Stats ( Strut cearler).

re type \*/

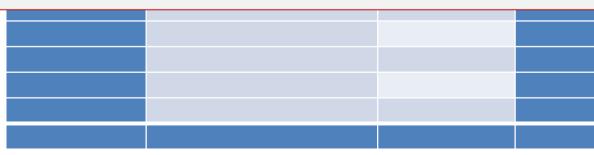
/\* ending ; \*/

## Structures in C

**END OF PART 1** 

- structures can be nested
- so that a field inside a structure can be another structure

```
397 - 398 - 399 -
1 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 1 1 0 1
400 - 401 - 402 -
0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1
403 - 404 - 405 -
```



```
struct name {
    char first[30];
    char last[30];
};

struct person {
    int age;
    float ppg;
    struct name title;
};
```

Address Label	Label	Address	Value

struct person boss;

	boss boss.age	400 - 403	
	boss.ppg	404 - 407	
boss.title.first	boss.title boss.title.first[0]	408 -	
	boss.title.first[1]	409 -	
	boss.title.first[2]	410 -	
	boss.title.first[3]	411 -	
	boss.title.first[4]	412 -	
	boss.title.first[5] – [31]	413 - 439	
boss.title.last	boss.title.last[0]	440 -	
	boss.title.last[1]	441 -	
	boss.title.last[2]	442 -	
	boss.title.last[3]	443 -	
	boss.title.last[4]	444 -	
	boss.title.last[5]	445 -	
	boss.title.last[6] – [31]	446 - 471	

Label

Address

Value

hoss. age.

struct person boss;

/\* fill in some values
 for the nested
 structure \*/

boss.age=40;

Address Label	Label	Address	Value
	boss boss.age	400 - 403	40
	boss.ppg	404 - 407	
boss.title.first	boss.title boss.title.first[0]	408 -	
	boss.title.first[1]	409 -	
	boss.title.first[2]	410 -	
	boss.title.first[3]	411 -	
	boss.title.first[4]	412 -	
	boss.title.first[5] – [31]	413 - 439	
boss.title.last	boss.title.last[0]	440 -	
	boss.title.last[1]	441 -	
	boss.title.last[2]	442 -	
	boss.title.last[3]	443 -	
	boss.title.last[4]	444 -	
	boss.title.last[5]	445 -	
	boss.title.last[6] – [31]	446 - 471	

struct person boss;

/\* fill in some values
 for the nested
 structure \*/

boss.age=40;
boss.ppg=0.1;

Address Label	Label	Address	Value
	boss boss.age	400 - 403	40
	boss.ppg	404 - 407	0.1
boss.title.first	boss.title boss.title.first[0]	408 -	
	boss.title.first[1]	409 -	
	boss.title.first[2]	410 -	
	boss.title.first[3]	411 -	
	boss.title.first[4]	412 -	
	boss.title.first[5] – [31]	413 - 439	
boss.title.last	boss.title.last[0]	440 -	
	boss.title.last[1]	441 -	
	boss.title.last[2]	442 -	
	boss.title.last[3]	443 -	
	boss.title.last[4]	444 -	
	boss.title.last[5]	445 -	
	boss.title.last[6] – [31]	446 - 471	

```
struct name {
                        first[30];
            char
                        last[30];
            char
            } ;
struct person {
            int
                        age;
            float
                        ppg;
            struct name title;
            } ;
struct person boss;
/* fill in some values
   for the nested
   structure */
boss.age=40;
boss.ppg=0.1;
strcpy (boss.title.first, "Dea
```



boss

boss.age

Label

	boss.ppg	404 - 407	0.1
e.first	boss.title boss.title.first[0]	408 -	D
	boss.title.first[1]	409 -	е
ean");	boss.title.first[2]	410 -	a
	boss.title.first[3]	411 -	n
	boss.title.first[4]	412 -	10
	boss.title.first[5] – [31]	413 - 439	
boss.title.last	boss.title.last[0]	440 -	
	boss.title.last[1]	441 -	
	boss.title.last[2]	442 -	
	boss.title.last[3]	443 -	
	boss.title.last[4]	444 -	
	boss.title.last[5]	445 -	
	boss.title.last[6] – [31]	446 - 471	

Address

400 - 403

Value

40

```
struct name {
           char first[30];
                      last[30];
           char
           } ;
struct person {
           int
                      age;
           float
                      ppg;
           struct name title;
           };
struct person boss;
/* fill in some values
  for the nested
   structure */
boss.age=40;
boss.ppg=0.1;
strcpy(boss.title.first,"Dean")
strcpy(boss.title.last,"Smith")
```

boss.ppg 404 - 407 0.1  e.first boss.title boss.title.first[0] 408 - D	
boss.title.first[1] 409 - e	
boss.title.first[2] 410 - a	
boss.title.first[3] 411 - n	
boss.title.first[4] 412 - \0	
boss.title.first[5] – [31] 413 - 439	
boss.title.last boss.title.last[0] 440 - 5	
boss.title.last[1] 441 - m	
boss.title.last[2] 442 - i	
boss.title.last[3] 443 -	
boss.title.last[4] 444 -	
boss.title.last[5] 445 -	
boss.title.last[6] – [31] 446 - 471	

Address

Value

Label

Label

## Structures in C

**END OF PART 2** 

#### **POINTERS and STRUCTURES**

- can be stored as an address in a pointer variable
- just like any other variable

```
struct fraction {
    int wP;
    int fP;
};

struct fraction f[3], *g;
```

399 -

Address Label	Label	Address	Value
<del>f</del>	f[0] f[0]. wP	400 - 403	
	f[0]. fP	404 - 407	
	f[1] f[1]. wP	408 - 411	
	f[1]. fP	412 - 415	
	f[2] f[2]. wP	416 - 419	
	f[2]. fP	420 - 423	
	g	424 - 427	

## **POINTERS and STRUCTURES**

- can be stored as an address in a pointer variable
- just like any other variable

	just like ally o	uioi vaiic	1010				0 0	0 1	1 0	1 1 0	0 1 1	1 0	0 1	1 0	1 1	1
						403 -			404	-		405				
st	ruct fraction	{				0 1 0	1 0	1 1			1 0 1			0 0	1 1	0
		•	D -		5	406 -			407	′- 110		408 -				•
		int	wP;	3	412	1 1 1 409 -	0 0	1 1	410		0 1 1	411		1 0	0 1	U
		int	fP;	- 1. 6		0 0 1	0 0	0 1			0 1 1			1 0	1 1	1
		};		- you	. 3 1/2	412 -			413	<b>3</b> -		414	-			
		1/		2/0		101	1 1	0 0	1 1	0 1 1	100	11	0 0	0 1	1 0	1
				> 1/1s	A/200	415 -			416			417 -				_
st	ruct fraction	f[3]	fg;)	$\times$ $\%$	22 VI	1 1·1 418-	0 0	1 1	0 1 419	110	0 1 1	L O O 420 -		1 0	0 1	0
						0 1 0	1 0	1 1			1 0 1			0 0	1 1	0
			*			421 -	- 0		422			423 -		•		
1	[0].wP = 3;			Label						Add	ress			Va	alue	2
1	[0].fP = 7;															_
	g = &(f[0]);	A.M.	241													
_	<i>σ</i> - α(τ[σ]),		24/	f[0]	f[0]. w	'P				40	00 -	403	3		3	
				f[0]. f	Р					40	04 -	407	7		7	
				f[1]	f[1]. w	P				40	08 -	411	L			
				f[1]. f	Р					4:	12 -	415	5			
				f[2]	f[2]. w	P				4:	16 -	419	)			
				f[2]. f	Р					42	20 -	423	3			
				g						42	24 -	427	7	4	<b>400</b>	

**399** -

402 -

110011011100110001101

401 -

#### **POINTERS and STRUCTURES**

- can be stored as an address in a pointer variable

int wP;

fP;

int

**}**;

- just like any other variable

struct fraction {

f[0] f[0]

q =

g++;

```
101 - 402 - 403 - 404 - 405 - 407 - 408 - 411 - 412 - 413 - 414 - 417 - 411 - 412 - 413 - 414 - 417 - 411 - 412 - 413 - 414 - 417 - 411 - 412 - 413 - 414 - 417 - 411 - 412 - 413 - 417 - 417 - 418 - 419 - 420 - 410 - 421 - 422 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 - 423 -
```

110011011100110001101

ct fraction f	[3], *g;		418 - 419	1 1 0 0 1 1 0 0 0 1 3 - 420 - 0 1 0 1 0 1 1 0 1 0 0 2 - 423 -	
.wP = 3; $.fP = 7;$	Address Label	Label		Address	Value
&(f[0]);	f	f[0] f[0]. v	vΡ	400 - 403	3
		f[0]. fP		404 - 407	7
y now		f[1] f[1]. v	vΡ	408 - 411	
contains		f[1]. fP		412 - 415	
100 + 8		f[2] f[2]. v	vΡ	416 - 419	
108 */		f[2]. fP		420 - 423	
		g		424 - 427	408

## **POINTERS and STRUCTURES**

- can be stored as an address in a pointer variable
- just like any other variable

Just like ally confer variable	0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1
	403 - 404 - 405 -
struct fraction {	010101111010101101000110
` '	406 - 407 - 408 -
int wP;	111001101110011000110010
int fP;	409 - 410 - 411 -
TIIL IF,	00100011011001110111
};	412 - 413 - 414 -
J <b>/</b>	10111001101110011001101
	415 - 416 - 417 -
struct fraction f[3], *g;	111001101110011000110010
scruce fraction f[5], g,	418 - 419 - 420 -
	010101111010101101000110
6101 5 0	421 - 422 - 423 -
f[0].wP= 3; Address Label	Label Address Value

399 -

402 -

408

110011011100110001101

401 -

f[0].wP= 3; f[0].fP= 7;	Address Label	Label	Address	V
g = &(f[0]);	f	f[0] f[0]. wP	400 - 403	
q++;		f[0]. fP	404 - 407	
9'',		f[1] f[1]. wP	408 - 411	
(*g).wP = 5;		f[1]. fP	412 - 415	1
g->fP = 11;		f[2] f[2]. wP	416 - 419	
		f[2]. fP	420 - 423	
		g	424 - 427	

#### **POIN** - two ways of specifying the field to use. - ca - jus what we have already seen: (\*g).wP = 5;struct // \* (at the address) of g go to the field at wP C also has a direction symbol struct q->fP = 11;f[0].wi - which can be used on addresses of structures f[0].fi go to the **structure** pointed to by **g** q = & (go to the **fP variable** of that structure. q++; f[1]. fP 412 - 415 11 (\*q).wP = 5;f[2] f[2]. wP 416 - 419 q->fP = 11;f[2]. fP 420 - 423 408 424 - 427

## **POINTERS and STRUCTURES**

397 -	398 -	399 -
101110	011011100	110001101
400 -	401 -	402 -
001000	110110011	100110111
403 -	404 -	405 -
010101	111010101	101000110
406 -	407 -	408 -
111001	101110011	000110010
409 -	410 -	411 -
001000	110110011	100110111
412 -	413 -	414 -
101110	011011100	110001101
415 -	416 -	417 -
111001	101110011	000110010
418 -	419 -	420 -
010101	111010101	101000110
421 -	422 -	423 -

Address Label	Label	Address	Value
	fract.wP	400 - 403	
	fract.fP	404 - 407	

## **POINTERS and STRUCTURES**

397 -	398 -	399 -
101110	001101110	0110001101
400 -	401 -	402 -
001000	0 1 1 0 1 1 0 0 1	11001111
403 -	404 -	405 -
010101	1 1 1 1 0 1 0 1 0	1 1 0 1 0 0 0 1 1 0
406 -	407 -	408 -
111001	1 1 0 1 1 1 0 0 1	1000110010
409 -	410 -	411 -
001000	011011001	1 1 0 0 1 1 0 1 1 1
412 -	413 -	414 -
101110	001101110	0 1 1 0 0 0 1 1 0 1
415 -	416 -	417 -
111001	110111001	1000110010
418 -	419 -	420 -
010101	1 1 1 1 0 1 0 1 0	1101000110
421 -	422 -	423 -

	421 -	422 - 423 -	
Address Label	Label	Address	Value
	fract.wP	400 - 403	7
	fract.fP	404 - 407	3

## **POINTERS and STRUCTURES**

```
struct fraction {
    int wP;
    int fP;
} fract;
```

397 -	398 -	399 -
101110	001101110	0110001101
400 -	401 -	402 -
001000	11011001	1100110111
403 -	404 -	405 -
010101	111101010:	1101000110
406 -	407 -	408 -
111001	110111001	1000110010
409 -	410 -	411 -
001000	11011001	1100110111
412 -	413 -	414 -
101110	001101110	0110001101
415 -	416 -	417 -
111001	110111001	1000110010
418 -	419 -	420 -
010101	111101010:	1101000110
421 -	422 -	423 -

Address Label	Label	Address	Value
	fract.wP	400 - 403	
	fract.fP	404 - 407	

#### **POINTERS and STRUCTURES**

```
397 -
         398 -
                  399 -
101110011011100110001101
         401 -
001000110110111001110111
         404 -
01010111101010110101000110
11100110111001100011001
001000110110011100111111
101110011011100110001101
        416 -
                 417 -
11100110111001100011001
01010111101010110101000110
        422 -
                 423 -
```

Address	Label	Address	Value
Label			
	fract.wP	400 - 403	7
	fract.fP	404 - 407	3

## **POINTERS and STRUCTURES**

- a structure can be declared.
- and/or declared later

397 -	398 -	<b>399</b> -
10111	00110111	00110001101
400 -	401 -	402 -
00100	01101100	11100110111
403 -	404 -	405 -
01010	11110101	01101000110
406 -	407 -	408 -
11100	11011100	11000110010
409 -	410 -	411 -
00100	0 1 1 0 1 1 0 0	11100110111
412 -	413 -	414 -
10111	00110111	00110001101
415 -	416 -	417 -
11100	11011100	11000110010
418 -	419 -	420 -
01010	11110101	01101000110
421 -	422 -	423 -

Address Label	Label	72.	Address		Value
	fract.wP		400 - 40	03	
	fract.fP		404 - 40	07	
	newNum.wP		408 - 41	l1	
	newNum.fP		412 - 41	L5	

### **POINTERS and STRUCTURES**

- a structure can be declared.

```
struct fraction {
                   int wP;
                   int fP;
                 } fract;
struct fraction newNum;
fract.wp = 7;
fract.fP = 3;
```

newNum.wp = 2;newNum.fP = 6;

397 -398 -399 -101110011011100110001101 401 -0010001101100111001111 404 -01010111101010110101000110 11100110111001100011001 001000110110011100111 101110011011100110001101 416 -417 -11100110111001100011001 01010111101010110101000110 421 -422 -423 -

		721		723	
Address Label	Label		Address		Value
	fract.wP		400 - 4	.03	7
	fract.fP		404 - 4	.07	3
	newNum.wP		408 - 4	11	2
	newNum.fP		412 - 4	15	6

# **POINTERS and STRUCTURES**

- a structure can be declared.

```
typedef struct fraction {
        int wP;
        int fP;
    } fract;
```

### notice:

- the **typedef** does NOT allocate memory for the variable.
- **fract** is just the **alias** for this construct.

397 -	398 -	<b>399</b> -
1011	100110111	00110001101
400 -	401 -	402 -
0010	0 0 1 1 0 1 1 0 0	1110011111
403 -	404 -	405 -
0 1 0 1	0 1 1 1 1 0 1 0 1	0 1 1 0 1 0 0 0 1 1 0
406 -	407 -	408 -
1110	0 1 1 0 1 1 1 0 0	11000110010
409 -	410 -	411 -
0010	0 0 1 1 0 1 1 0 0	1110011111
412 -	413 -	414 -
1011	100110111	00110001101
415 -	416 -	417 -
1 1 1 0	011011100	11000110010
418 -	419 -	420 -
0 1 0 1	0 1 1 1 1 0 1 0 1	0 1 1 0 1 0 0 0 1 1 0
421 -	422 -	423 -

		421	422 -	423 -	
Address Label	Label		Address		Value
Labol					
Label					

# **POINTERS and STRUCTURES**

- a structure can be declared.

397 -	398 -	399 -
101110	0 1 1 0 1 1 1 0 0	110001101
400 -	401 -	402 -
001000	110110011	100110111
403 -	404 -	405 -
010101	1 1 1 0 1 0 1 0 1	101000110
406 -	407 -	408 -
111001	101110011	000110010
409 -	410 -	411 -
001000	1 1 0 1 1 0 0 1 1	100110111
412 -	413 -	414 -
101110	0 1 1 0 1 1 1 0 0	1 1 0 0 0 1 1 0 1
415 -	416 -	417 -
111001	101110011	000110010
418 -	419 -	420 -
010101	111010101	101000110
421 -	422 -	423 -

Address	Label	Address	Value
Label			
	first.wP	400 - 403	
	first.fP	404 - 407	
	second.wP	408 - 411	
	second.fP	412 - 415	

### **POINTERS and STRUCTURES**

- a structure can be declared.

101110011011100110001101 0010001101100111001111111 typedef struct fraction { 01010111101010110101000110 int wP; 11100110111001100011001 int fP; 0010001101100111001110111 } fract; 101110011011100110001101 fract first, second; 2 Struct 7 416 417 416 417 419 419 420 420 418 419 420first.wp = 7;**Address** Label **Address** Value first.fP = 3;Label 400 - 403 first.wP second.wp = 2;first.fP 404 - 407 second.fP = 6;second.wP 408 - 411 second.fP 412 - 415

397 -

399 -

# Structures in C

**END OF PART 3** 

### **POINTERS and STRUCTURES**

- a pointer variable can also be used

to **DYNAMICALLY** allocate memory.

Address Label	Label	Address	Value
f	f[0] f[0]. wP	400 - 403	
	f[0]. fP	404 - 407	
	f[1] f[1]. wP	408 - 411	
	f[1]. fP	412 - 415	
	f[2] f[2]. wP	416 - 419	
	f[2]. fP	420 - 423	
	g	424 - 427	

#### **POINTERS and STRUCTURES** - a pointer variable can also be used 01110011011100110001101 to **DYNAMICALLY** allocate memory. 01000110110011100111 struct fraction { 1000110 int wP; 0110010 int fP; 0110111 0001101 struct fraction f[3], \*q; 0110010 g = (struct fraction \*)malloc( sizeof(struct fraction)); 1000110 423 -Address Label **Address** Value Label f[0]. wP 400 - 403 f[0] f[0]. fP 404 - 407 f[1] f[1]. wP 408 - 411 f[1]. fP 412 - 415 f[2] f[2]. wP 416 - 419 f[2]. fP 420 - 423

g

{DM}

424 - 427

10100 - 10107

10100

# POINTERS and STRUCTURES - a pointer variable can also be used to **DYNAMICALLY** allocate memory. struct fr g = (struct fraction \*)malloc( sizeof(struct fraction)); - what is happening here? struct fr - a struct of type **fraction** is how many bytes in size ? q = (stru - 8 bytes (2 integer values x 4 bytes) - so: ( sizeof( struct fraction ) ) will return 8 bytes of memory - malloc() returns a pointer of type void UNLESS specifically type cast as above - this tells the program the pointer references a memory location of a structure of the type **fraction** TOTOO

{DM}

10100 - 10107

#### POINTERS and STRUCTURES - a pointer variable can also be used 0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1 to **DYNAMICALLY** allocate memory. 01000110110011100111 struct fraction { 1000110 int wP; 0110010 int fP; 0110111 0001101 struct fraction f[3], \*q; 0110010 g = (struct fraction \*)malloc( sizeof(struct fraction)); 1000110 Address Label Address Value Label $q \rightarrow wP = 6$ ; (\*g).fP=1; f[0] f[0]. wP 400 - 403 f[0]. fP 404 - 407 f[1] f[1]. wP 408 - 411 f[1]. fP 412 - 415 f[2] f[2]. wP 416 - 419 f[2]. fP 420 - 423 424 - 427 10100 g 10100 - 10103 g.wP 10104 - 10107

# **POINTERS and STRUCTURES**

- a pointer variable can also be used to **DYNAMICALLY** allocate memory.

7- 398- 399-0 1 1 1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 1 1 0 1 0- 401- 402-

```
struct fr

struct fr

g = (struct)

g->wP=6;
(*g).fP=
```

g = (struct fraction \*)malloc( sizeof(struct fraction));

malloc allocated structure size (8 bytes) of memory for use
malloc( sizeof(struct fraction))

- g (as a pointer variable)

holds the address of this memory location

- g indirection allows the use of the memory

SO: as stated:

typecasting (struct fraction \*) allows the O/S to know how to deal with this memory allocation

# Structures in C

**END OF PART 4** 

# CS 2211 Systems Programming

Part Ten: Unions

		397 - 398 - 399 -	
		101110011011100110001101	l
		400 - 401 - 402 -	
		00100011011001110011111111	1
		403 - 404 - 405 -	
		010101111010101101000110	)
		406 - 407 - 408 -	
Address Label	Label	11100110111001100011001	)
		409 - 410 - 411 -	
		0010001101100111001111111111	l
		412 - 413 - 414 -	
		101110011011100110001101	1
		415 - 416 - 417 -	
		11100110111001100011001	)
		418 - 419 - 420 -	
		01010111101010110101000110	)
		421 - 422 - 423 -	
		10111001101110011000110	1

- a construct to group together dissimilar

```
variables under one name
                                      101110011011100110001101
                                      0010001101100111001110111
                                      01010111101010110101000110
                 Address Label
                            Label
                                      111001101110011000110010
/* "demographics" is name for union type */
                                                              1 1
union demographics {
                                                              0 1
       int
                  /* 1st field is person's age */
              age
       float salary; /* 2nd field is monthly pay */
                                                              1 0
       double emplevel; /* 3rd field is rating scale */
                                                              1 0
       char Owing; /* 4th field is office wing */
                                                              0 1
                           /* ending ; */
       };
/* means end of union type definition */
```

- a construct to group together dissimilar

```
variables under one name
                                                       399 -
                                       101110011011100110001101
                                       0010001101100111001110111
                                       01010111101010110101000110
                  Address Label
                                       111001101110011000110010
/* "demographics" is name for union type */
                                                              1 1
union demographics {
                                                              0 1
                    /* 1st field is person's age */
       int
              age
       float salary; /* 2nd field is monthly pay */
                                                              1 0
       double emplevel; /* 3rd field is rating scale */
                                                              1 0
       char Owing; /* 4th field is office wing */
                                                              0 1
                           /* ending ; */
       };
/* means end of union type definition */
```

#### NOTE:

This code does NOT create a variable (similar to Class declaration in OOP)

- grandfather of this concept
- this is ONLY a template for a variable declaration.
- this creates a **USER DEFINED** variable type

- a construct to group together dissimilar

```
variables under one name
                                        101110011011100110001101
                                        0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 1 0 1 1 1
/* "demographics" is name for union type */
                                                               1 1 0
union demographics {
                                                               0 1 0
                   /* 1st field is person's age */
       int
               age
       float salary; /* 2nd field is monthly pay */
                                                               111
       double emplevel; /* 3rd field is rating scale */
                                                               1 0 1
       char Owing; /* 4th field is office wing */
                                                               0 1 0
                          /* ending ; */
       };
                                                               1 1 0
/* means end of union type definition */
                                                               101
int i;
union demographics employee;
```

creates an integer variable (space to store an integer) i - and creates a demographic variable (space to store ONLY the largest of the variables in the UNION [in this case 8 bytes]) employee.

Address Label	Label	Address	Value

同品是12-12小块内存

Value

### **MEMORY MAPPING - UNIONS**

employee

employee.age

employee.salary

employee.Owing

employee.emplevel

400 - 403

400 - 403

400 - 407

400

**Address** 

memory 400 to 407

union demographics employee; Address Label

the union employee takes up (requires)8 bytes of data

#### note:

employee.salary
employee.age
employee.emplevel
employee.Owing
all returns the address 400

union demographics employee;

employee.age = 24;

Address Label	Label	Address	Value
employee	employee.age	400 - 403	24
	employee.salary	400 - 403	
	employee.emplevel	400 - 407	
	employee.Owing	400	

union demographics employee;

```
employee.age = 24;
employee.Owing = 'D';
```

Address Label	Label	Address	Value
employee	employee.age	400 - 403	
	employee.salary	400 - 403	
	employee.emplevel	400 - 407	
	employee.Owing	400	D

```
/* "demographics" is name for union type */
union demographics {
                                 /* 1st field is person's age */
             int.
             float salary;  /* 2nd field is monthly pay */
double emplevel;  /* 3rd field is rating scale */
                                            /* 4th field is office wing */
             char Owing:
      CAUTION!
        employee.Owing = 'D';
unic
                                                                                                        Value
      assigned the sequence 0100 0100 to the memory location 400
emp1
emp1
       BUT !!! Address 401 and 402 and 403 are UNCHANGED
        1.) these held the values from the previous assignment of:
                   employee.age = 24;
     So the sequence that was:
      0000 0000 0000 0000 0000 0000 0001 1000
     Is now changed to
     0100 0100 0000 0000 0000 0000 0001 1000
     Now:
              printf("age = %d", employee.age);
     produces:
              age = 570425356
```

Ato 82%.

实了上发行的不是

M(3)

### **MEMORY MAPPING - UNIONS**

```
/* "demographics" is name for union type */
union demographics {
              age /* 1st field is person's age */
       int.
       float salary; /* 2nd field is monthly pay */
       double emplevel; /* 3rd field is rating scale */
              Owing; /* 4th field is office wing */
       char
                                    /* ending ; */
       } employee;
                                                              alue
/* means end of union type definition */
union demographics employee;
```

```
/* "demographics" is name for union type */
typedef union demographics {
       int age /* 1st field is person's age */
       float salary; /* 2nd field is monthly pay */
       double emplevel; /* 3rd field is rating scale */
              Owing; /* 4th field is office wing */
       char
       dGraphics;
                                   /* ending ; */
                                                              alue
/* means end of union type definition */
dGraphics employee;
union demographics employee;
```

# **MEMORY MAPPING - STRUCTURE vs UNION**

	STRUCTURE	UNION
Keyword	The keyword <b>struct</b> is used to define a structure	The keyword union is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to the sum of sizes of its members.	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of union is equal to the size of largest member.
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated it shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

# Unions in C

**END OF PART 1**