

Deriving Types II

COMP 1002/1402

Nesting Structures

So far, only basic types appeared in **struct**

Q.

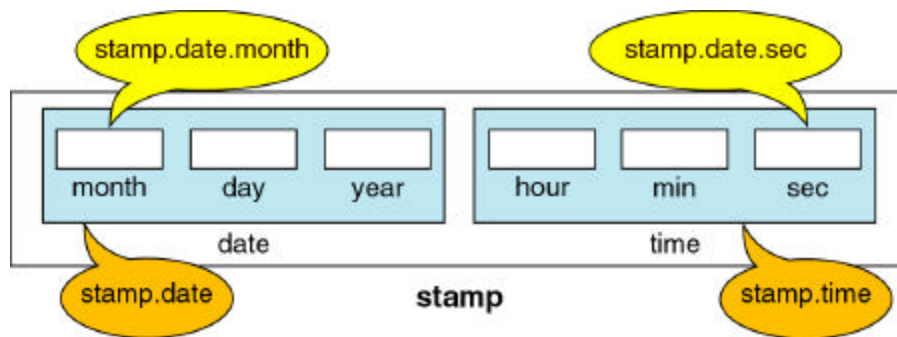
Why not defined types?

Ans.

No reason but simplicity of presentation

Nesting Structures

Define a **STAMP** to contain:
a **DATE** and a **TIME**



How to do it

Use one struct or type in the struct
(examples to follow)

Style: Declare every structure separately!

Necessity: Declare before use.

Bad Style

```
typedef struct {  
    struct {  
        int month;  
        int day;  
        int year;  
    } date;  
    struct {  
        int hour;  
        int min;  
        int sec;  
    } time;  
} STAMP;  
  
STAMP aStamp;
```

Good Style

```
typedef struct {  
    int month;  
    int day;  
    int year;  
} DATE;  
typedef struct {  
    int hour;  
    int min;  
    int sec;  
} TIME;  
typedef struct {  
    DATE date;  
    TIME time;  
} STAMP;  
  
STAMP aStamp;
```

Referencing Nested Structures

```
aStamp  
aStamp.date  
aStamp.date.month  
aStamp.date.day  
aStamp.date.year  
aStamp.time  
aStamp.time.hour  
aStamp.time.min  
aStamp.time.sec
```

Nested Structure Initialization

Initialize each structure with:

```
Nested {} :  
STAMP aStamp = {{05,10,1936},{23,45,00}};
```

Or predefined variables:

```
DATE aDate = {05,10,1936};  
TIME aTime = {23,45,00};  
STAMP aStamp = {aDate, aTime};
```

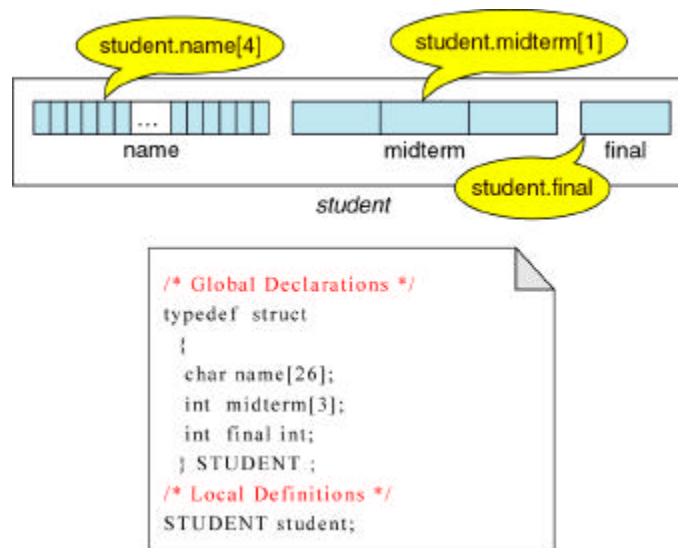
Arrays in Structures

Defined like any other element

Accessed with indices

Initialized like a nested (sub)structure

Arrays in Structures



Accessing Elements

```
STUDENT aStudent;
```

```
aStudent
```

```
aStudent.name
```

```
aStudent.name[1]
```

```
aStudent.midterm
```

```
aStudent.midterm[j]
```

```
aStudent.final
```

Accessing Elements

```
STUDENT *paStudent;
```

```
paStudent = &aStudent;
```

```
paStudent->name
```

```
paStudent->name[1]
```

```
paStudent->midterm
```

```
paStudent->midterm[j]
```

```
paStudent->final
```

Accessing pointers

```
STUDENT aStudent={"John Smith",{92,80,70},87};
```

```
int *pScores = student.midterm;
```

```
int totalScores = *pScores + *(pScores+1) +  
    *(pScores+2);
```

Pointers, Structures and Memory

Consider the DATE structure

Months should be strings!

Should we store the string in every month?

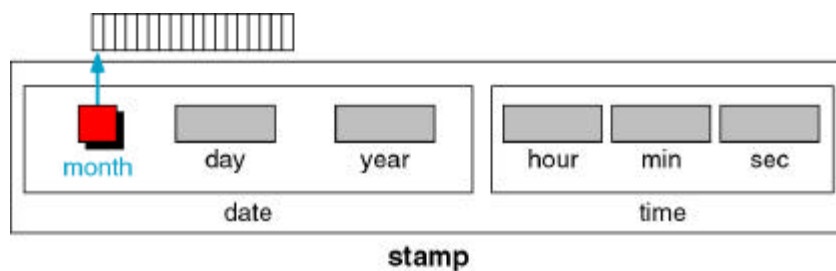
Store one pointer in every structure

The New Structure

```
typedef struct {  
    char *month;  
    int day;  
    int year;  
} DATE;
```

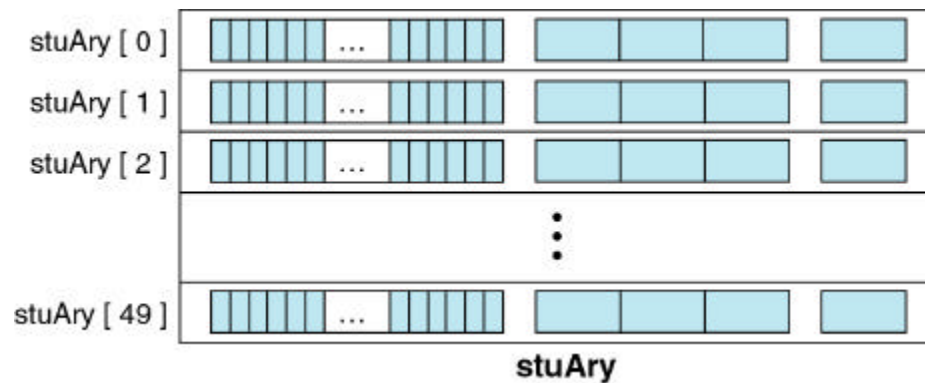
The New Structure

Every "December" points to the same spot!



Array of Structures

STUDENT stuAry[50];



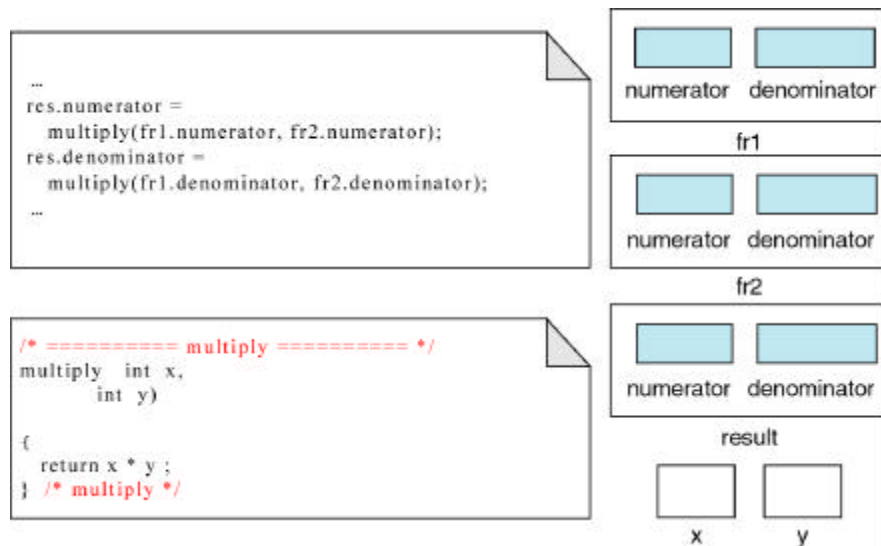
Array of Structures

```
int totScore = 0;
float average;
STUDENT *pStu;
STUDENT *pLastStu;
...
pLastStu = stuAry + 49;
for (pStu = stuAry; pStu <= pLastAry;pStu++)
    totScore += pStu->final;
average = totScore / 50.0;
```

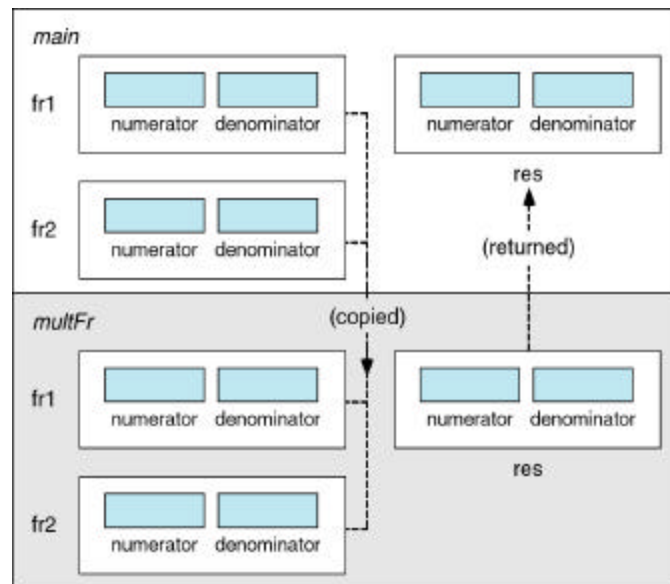
Structures and Functions

- Pass individual members (fields)
- Pass entire structure by value
- Pass address to structure

Passing Individual Members



Sending the Whole Structure



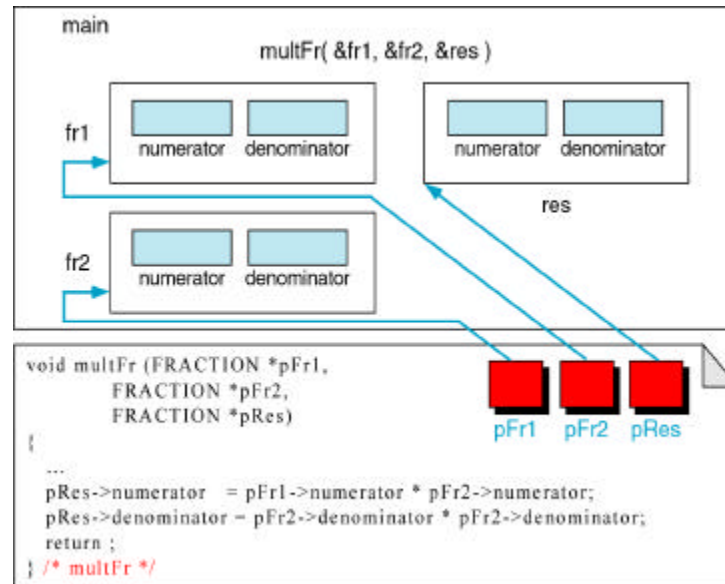
Careful Passing by Value!

Any pointers in the structure?

What about that new DATE class...

Pass a DATE to a function by value,
In the function change the contents of month,
What happens?

Passing Whole Structures



Unions

union like **enum** & **struct** allow:

- A single variable
- Multiple variables
- New type definition

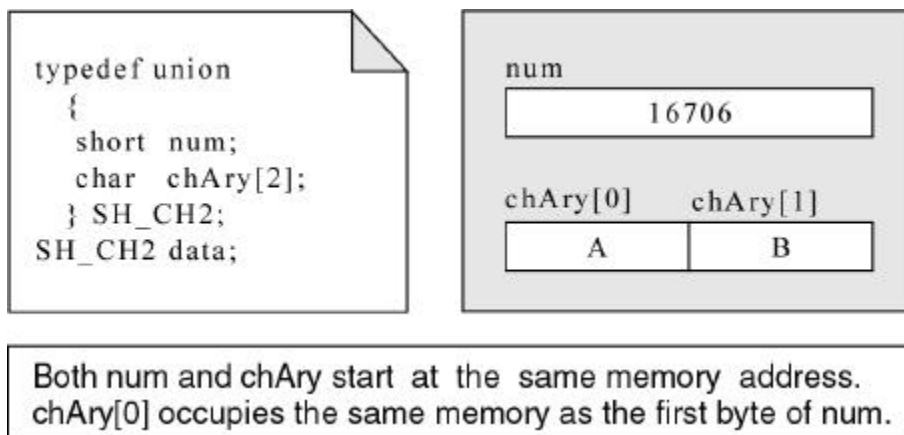
Union

A union variable allows:
more than one type of data
to occupy its memory!

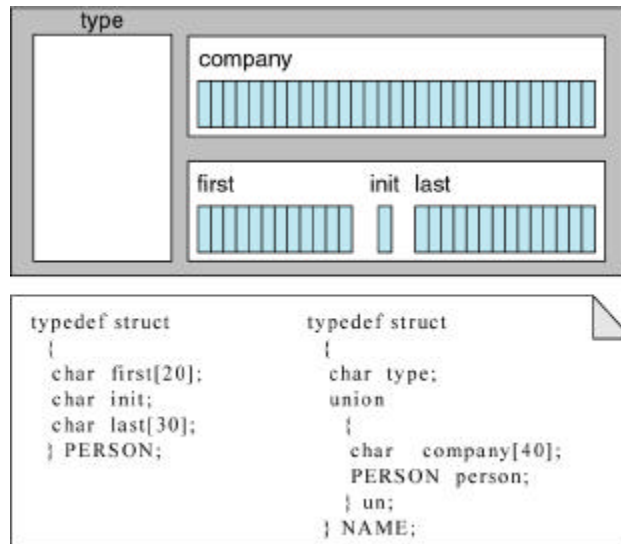
Big enough for the largest of them.

Accessed with . Operator
(**data.num** or **data.chAry[0]**)

union



Unions in Structures



Initializing Unions

Only the first type declared in union!

```
typedef union {
    short num;
    char ch[2];
} SH_CH2;
```

```
SH_CH2 data = 16706;
printf("%d\n%c\n%c\n", num, ch[0], ch[1]);
```

Little Endian, Big Endian

Q. What does this code produce?

```
SH_CH2 data = 16706;  
printf("%d\n%c\n%c\n", num, ch[0], ch[1]);
```

Ans. Two possible outputs!

16706	16706
A	B
B	A

Why?

$$(16706)_{10} = (0100\ 0001\ 0100\ 0010)_2$$

Big Endian puts **msw** before **lsb**

Little Endian puts **lsb** before **msw**

(Most Significant Word & Least Significant Word)

Word is 2 bytes (usually)!