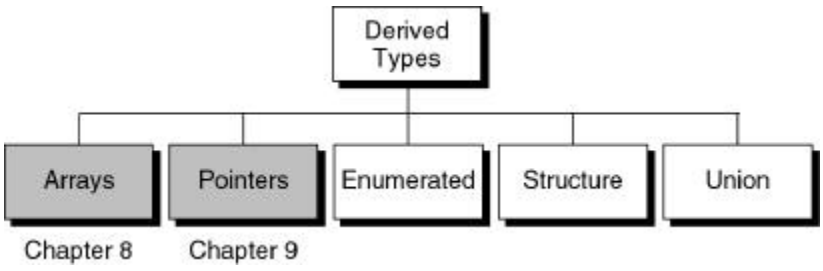
Deriving Types I

COMP 1002/1402

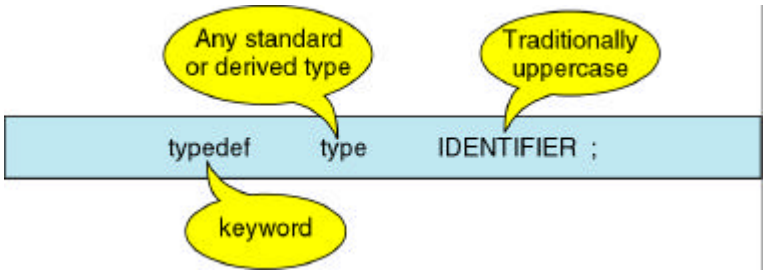
Derived Types



The Type Definition

`typedef`

– complex types defined as a different name



`typedef`

Examples:

```
typedef unsigned int ASIZE;

char *stringPtrAry[20];
```

Replace with:

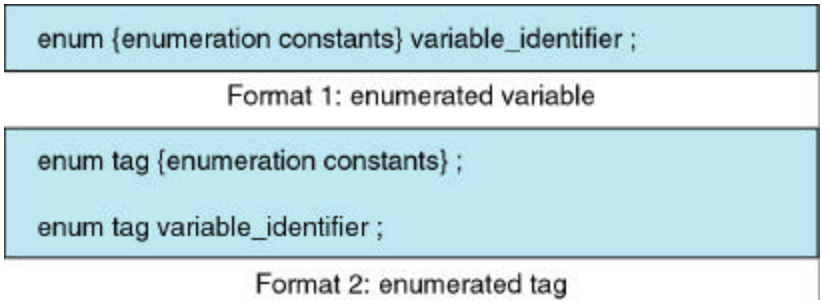
```
typedef char *STRING;
STRING stringPtrArray[20];
```

Enumerations

Identifies symbols with integers

Built on top of integers

`enum`



enum

Defining a single enumeration variable:

```
enum {sun, mon, tue, wed, thur, fri,
  sat} days;
days = mon;

if (days == mon)
  printf("I don’t like Mondays\n");
```

enum

Technically:

sun is **0**, **mon** is **1**, **tue** is **2**, …

Therefore:

```
  days = 1; /* is valid */
```

Careful! No range checking!

`enum`

Want different numbers?

```
enum {sun=1, mon=2, tue=3, wed=4,
  thur=5, fri=6, sat=7} days;
```

Or

```
enum {sun=1, mon, tue, wed, thur,
  fri, sat} days;
```

Enumerations

Caution:

ISO/ANSI C is not strict with **`enum`**

Don’t mix **`enum`** symbols with integer **variables**

Even if possible (depends on compiler)

Use:

`enum` symbols to replace constants!

enum

Defining Multiple enumeration variables with the same values!

```
enum DaysOfWeek {sun=1, mon, tue,
  wed, thur, fri, sat};
enum DaysOfWeek today;
enum DaysOfWeek yesterday;
```

`typedef` & `enum`

```
enum colors{red, white, blue, green, yellow};

enum colors aColor ;
```

Enumerated tag

```
typedef enum {red, white, blue, green, yellow} COLORS ;

COLORS aColor ;
```

Enumerated typedef

Structures

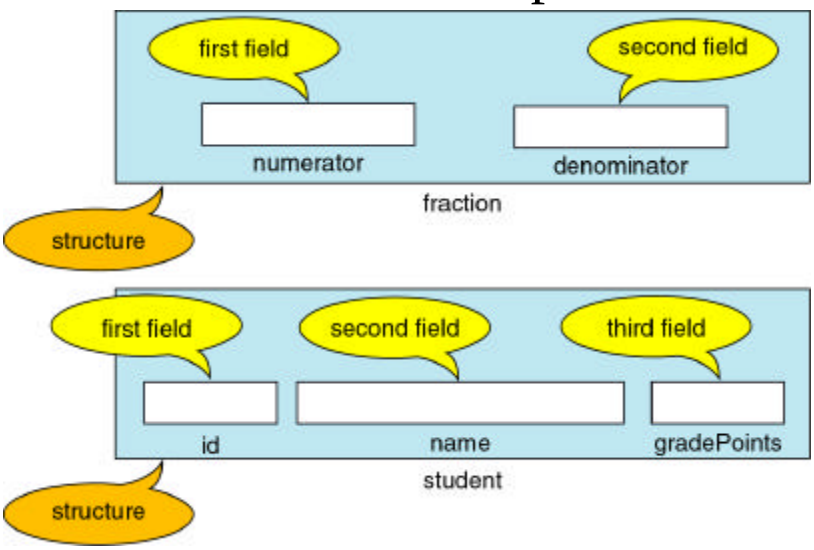
A structure is a collection of related elements

A **field** is the smallest element of named data that has meaning

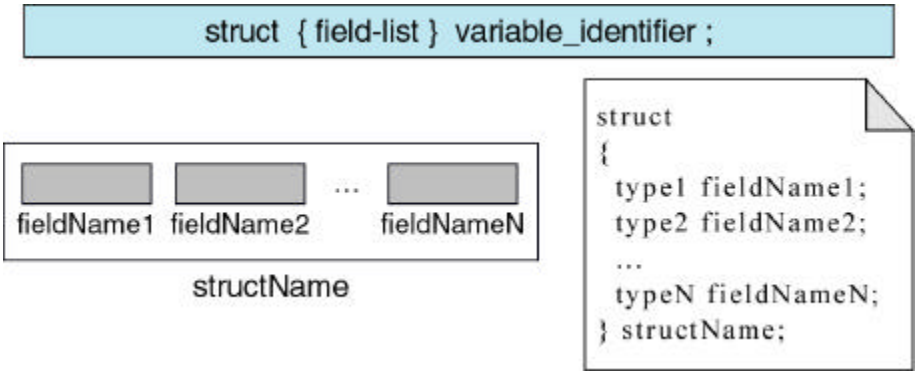
An array has elements of the same type

A structure can have elements of different types

Structure Examples



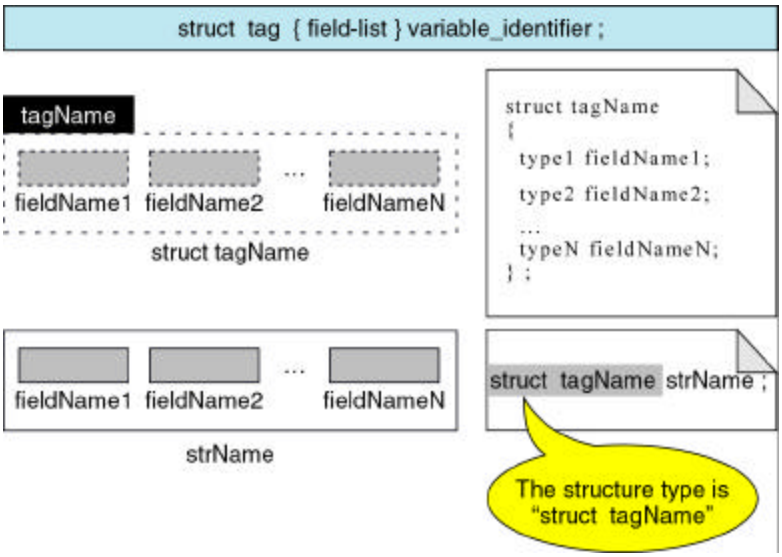
Defining a Single Structure Variable



`student` Variable

```
struct {
  char id[10];
  char name[26];
  int gradePoints;
} student;
```

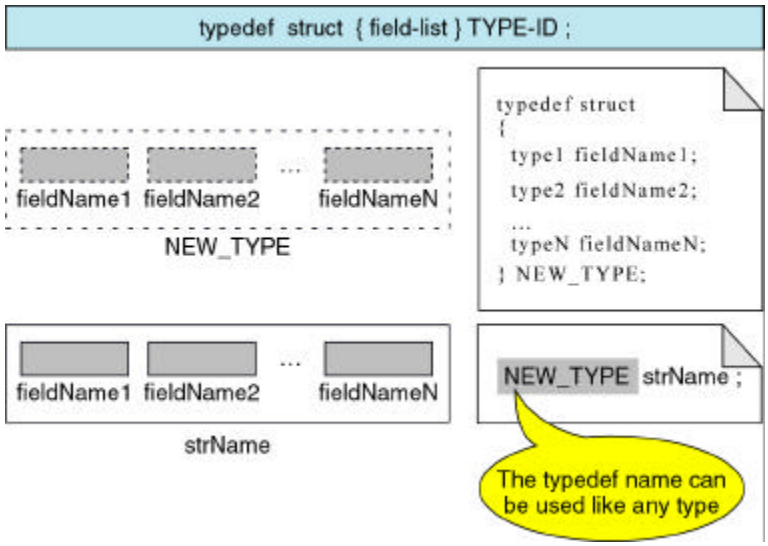
Multiple Structure Variables



`student` Variables

```
struct student {
  char id[10];
  char name[26];
  int gradePoints;
};

struct student aStudent;
struct student topStudent;

void printStudent (struct student Stu);
```

Defining Structure Types



STUDENT type

```
typedef struct {
  char id[10];
  char name[26];
  int gradePoints;
} STUDENT;

struct STUDENT aStudent;
struct STUDENT topStudent;

void printStudent (STUDENT Stu);
```

`struct` Summary

```
struct   {
           ...
         }  variable_identifier ;
```

structure variable

```
struct  tag
         {
           ...
         }  variable_identifier ;

struct  tag  variable_identifier ;
```

tagged structure

```
typedef  struct
         {
           ...
         }  TYPE_ID ;

TYPE_ID   variable_identifier ;
```

type-defined structure

Initializing0

```
typedef  struct
   {
   int   x;
   int   y;
   float t;
   char  u;
  } SAMPLE;
```

```
SAMPLE  sam1 = { 2,  5,  3.2,  'A' } ;
```

2	5	3.2	A
x	y	t	u

```
SAMPLE   sam2 = { 7, 3 } ;
```

filled with null ('\0')

7	3	0.0	\0
x	y	t	u

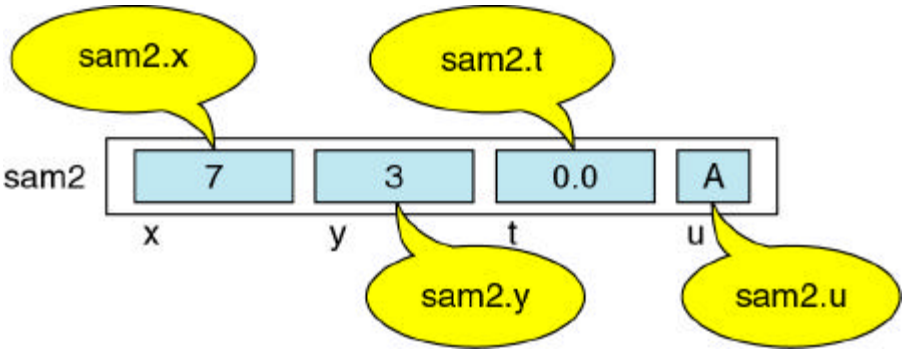
filled with float zero

Accessing

Given the **`aStudent`** variable, access its parts:

Use the **`.`** Operator

```
aStudent.id
aStudent.name
aStudent.gradePoints
```




```
if (sam2.u == 'A')
  sam2.x += sam2.y;
scanf("%d %d %f %c", &sam2.x,
  &sam2.y, &sam2.t, &sam2.u);
sam2.x++;
sam2.y++;
```

Comments on Precedence

- The `.` Operator has a very high precedence

```
sam.x++    ++sam.x    &sam.x
```

Are equivalent to:

```
(sam.x)++  ++(sam.x)  &(sam.x)
```

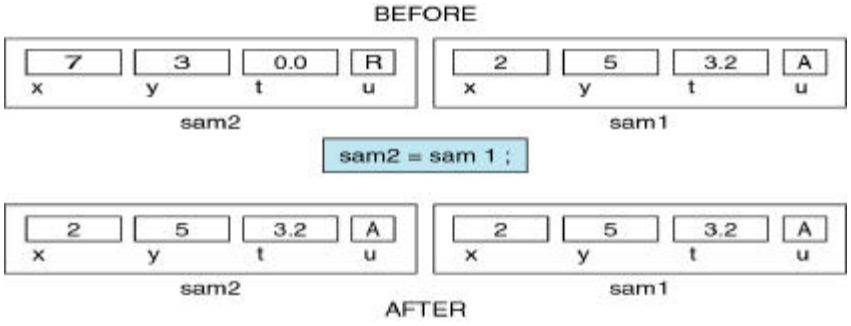
example

```
#include<stdio.h>

typedef struct {
  int numerator;
  int denominator;
} FRACTION;

/* write multiplyFraction (...)*/
```

Structure Operations

Assignment works on whole structures!

```
sam2 = sam1;
/* copies contents of sam1 into sam2!! */
```



Can’t do Comparison

== would compare all bits in structure

But some hardware require items start on word boundaries!

`floats` must start at address divisible by ?6?

`ints` must start at address divisible by ?4?

Thus structures aren't packed!

A Structure and its bytes:

a string bytes 0-24

bytes 25-29 (nothing but noise)

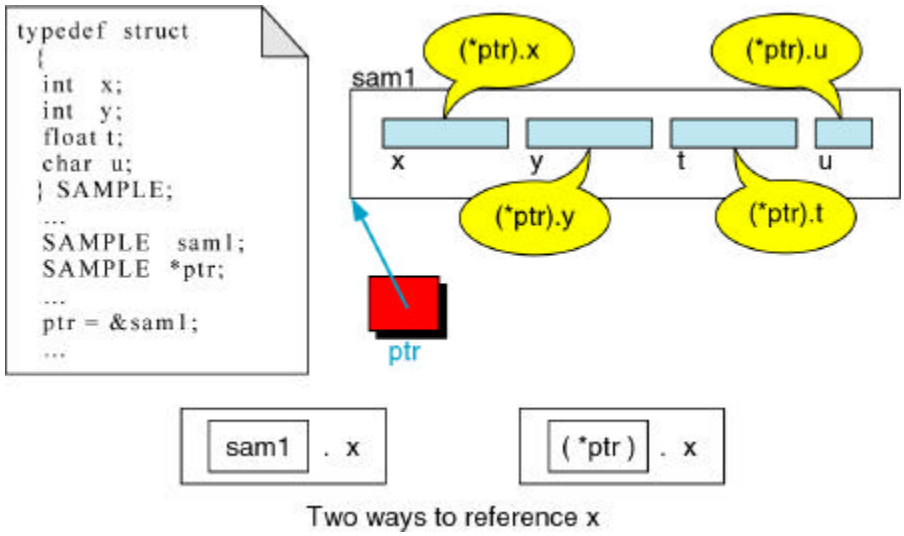
a float in bytes 30-35

a char in byte 36

bytes 37-39 (nothing but noise)

an int in bytes 40-43

Pointers to Structures



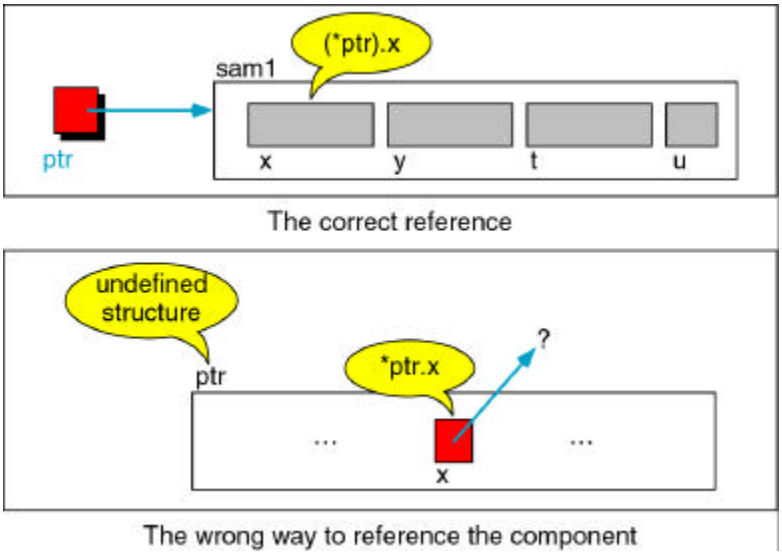
Two ways to reference x

Pointers to Structures

(*ptr).x
(*ptr).y
(*ptr).t
(*ptr).u

Don’t forget these parentheses!!!!
Common mistake and it’s deadly.

Pointers to Structures



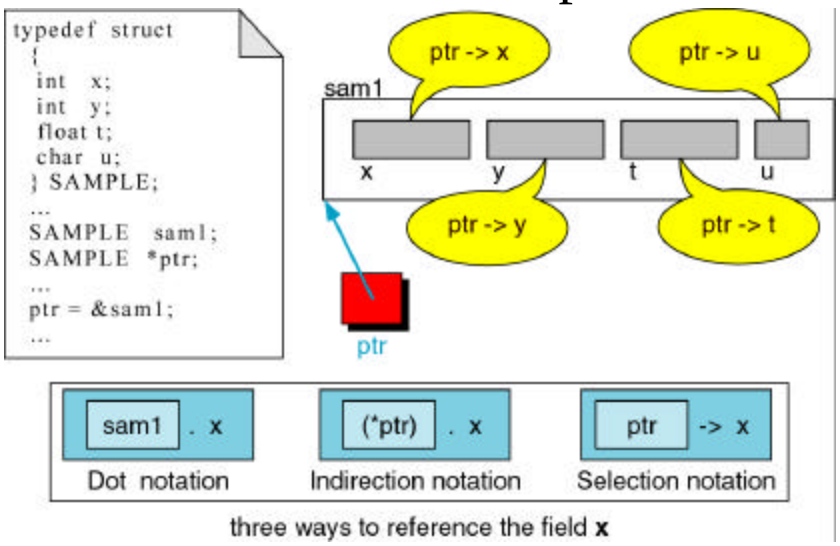
An Easier Way (selection)

`(*pointerName).fieldName`

Is the same as:

`pointerName->fieldName`

Pointer Selection Operator



Precedence of Selection

Precedence of **->**

Is identical to **.**

This implies they are equally high!

Example

```
typedef struct {
  int hr;
  int min;
  int sec;
} CLOCK;

void increment(CLOCK *clock);
void show(CLOCK *clock);
```