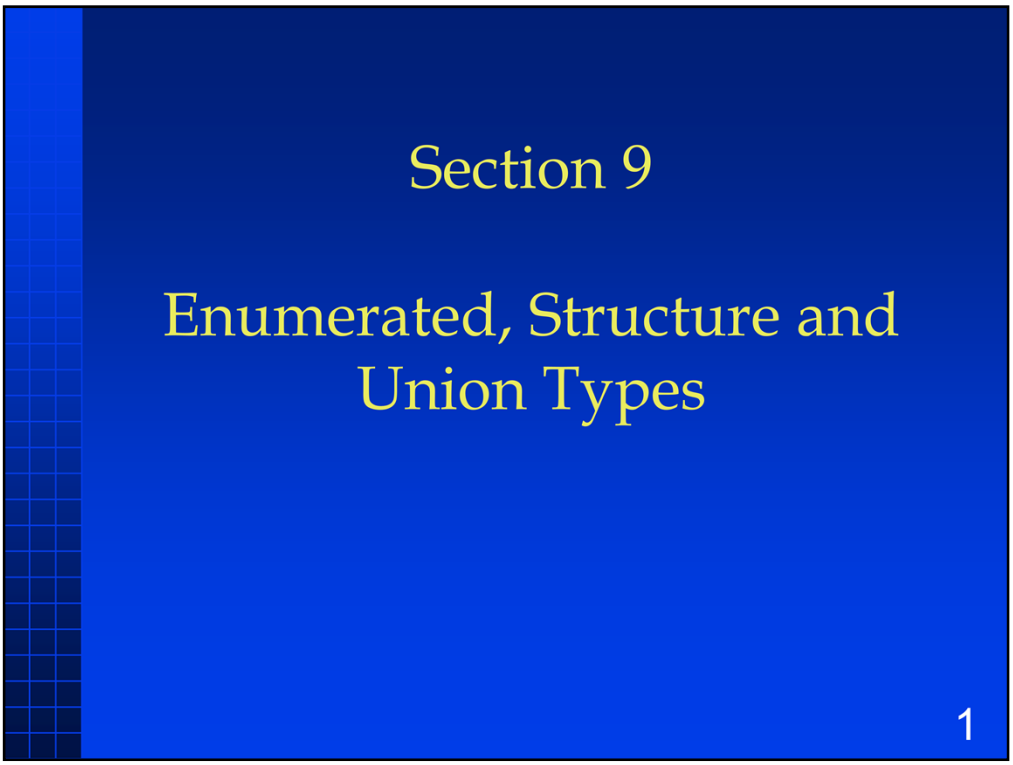
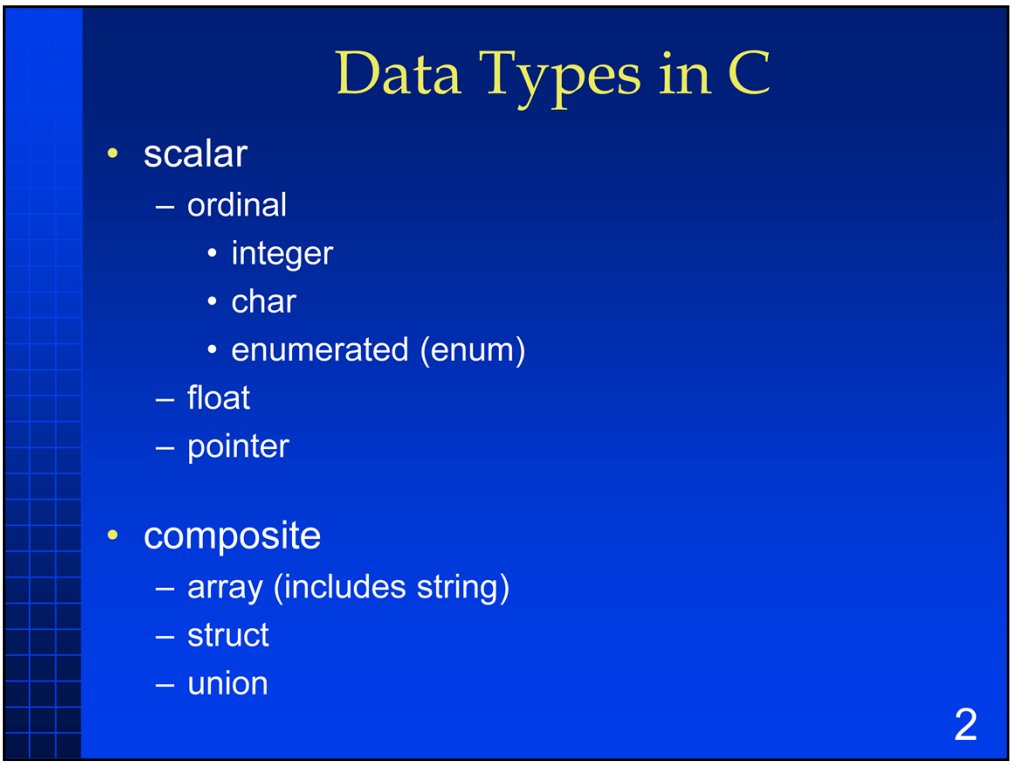


On the general topic of data structuring.

have seen builtin-types int, char float, pointers

composite type array, collection of elements of same type.

other structuring facilities, strongly influenced by Pascal and related. underlying principle is to model data according to application. could be business-rules, hardware abstractions, whatever.



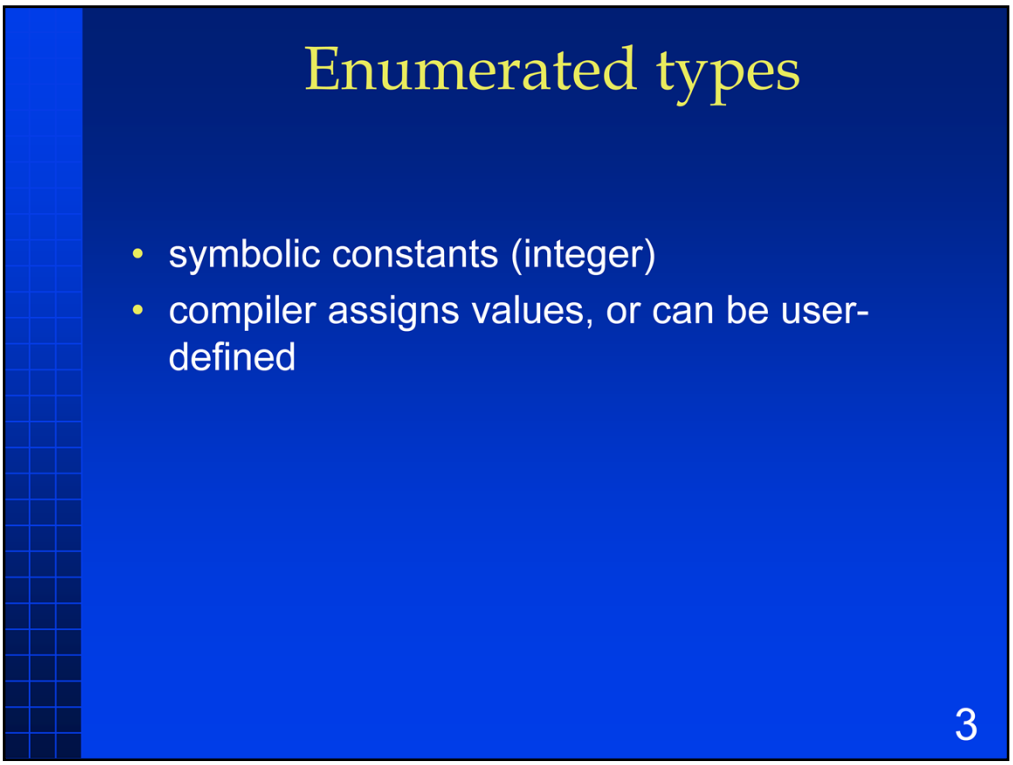
basic categories

we've seen most of the scalar, enum in a minute

for composite, will look at struct and union

struct: like records, dsect, structs in other langs

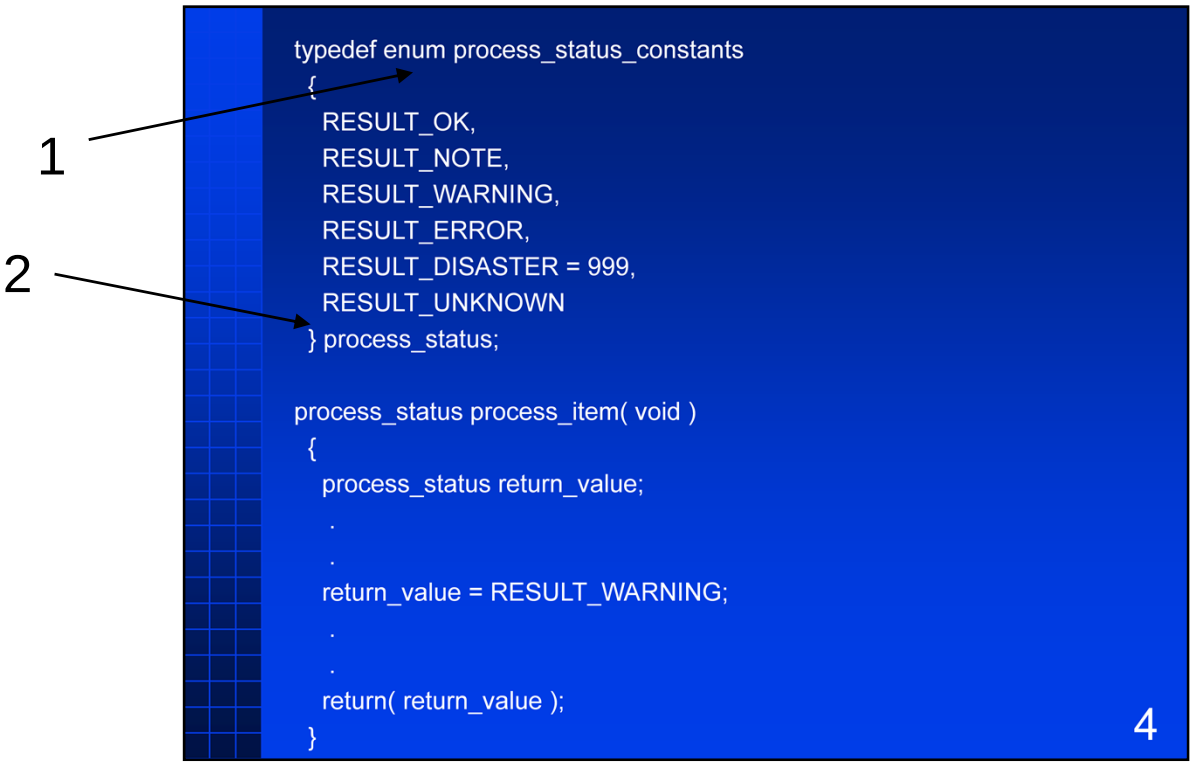
unions: several structs overlaid; assembler ORG, common block, redefines, variant



automatic way of generating a sequence of integer constants.

eg error codes, want to reference with symbolic names, don't care what the actual numbers are

very important for defining format interfaces eg Windows has thousands



eg.

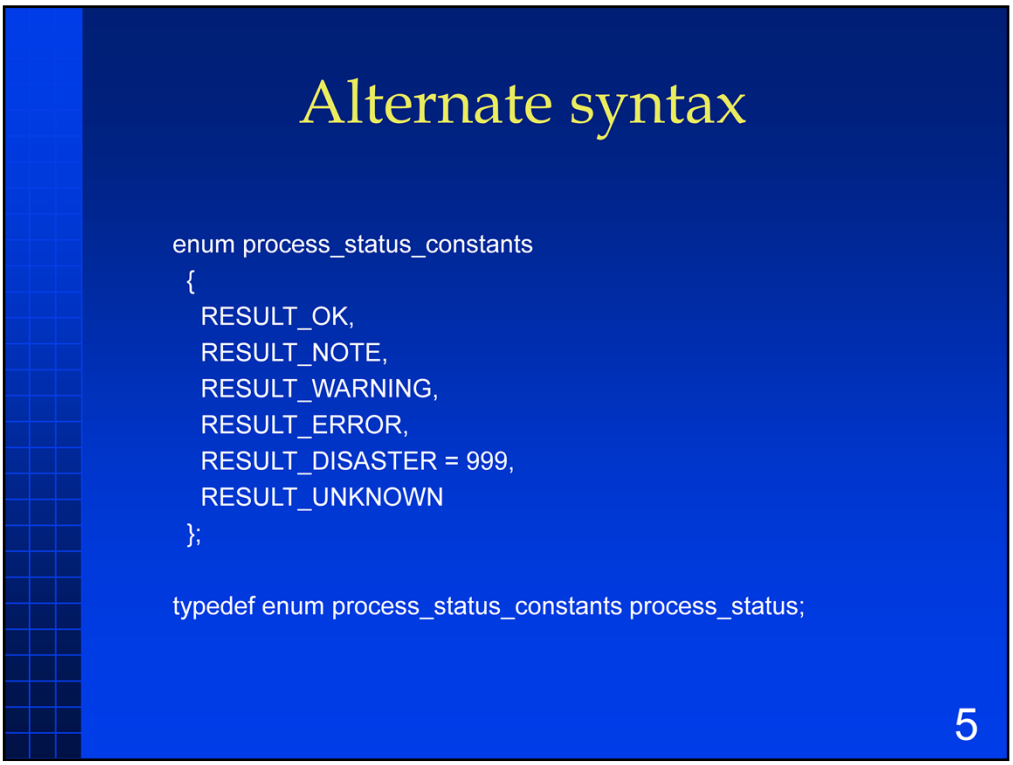
1) define a set of constants “enum process_status_constants”, given symbolic names, compiler generates integer values

if we want, can specify value

behave same as #define or const int,

2) also define a new typename that can be used in same circumstances as a built-in: var, function declarations and definitions.

Handy, too bad compiler doesnt enforce them (unlike Pascal): they’re all integers. Some compilers may issue warniongs, may be some other source-code management tools that can detect (eg lint)

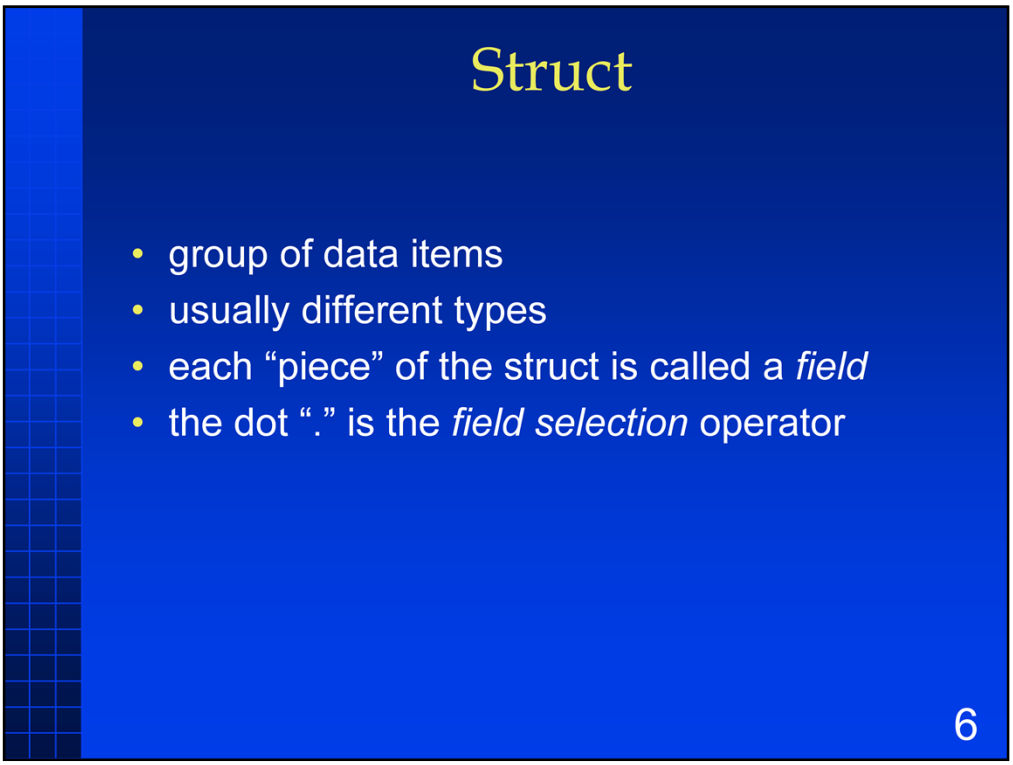


preceding syntax unclear that two distinct definitions taking place:

1) the declaration of the enumeration

2) the declaration of the new type

in general typedef just declares an alias for another type. eg want to rename int to integer, just say: typedef int integer



Struct is another example of a composite type.

arrays are homogeneous (all the same type); structs are heterogeneous (can be different types).

just a way of grouping things together and giving a common name

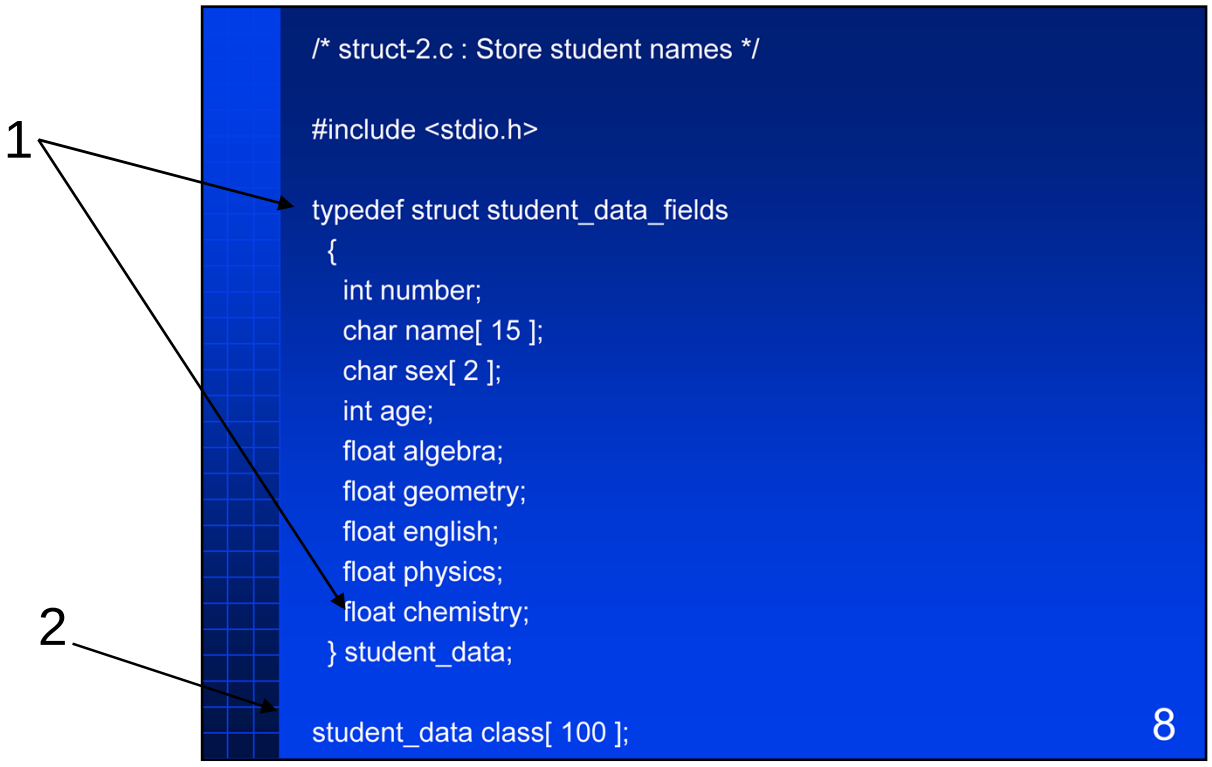
declares only the shape: storage definition occurs when struct used to defin a variable or function.

1) declaration of structure: three fields

Note style of syntax, no typedef in this case.

2) definition of variable; since no typedef, use struct-name directly.

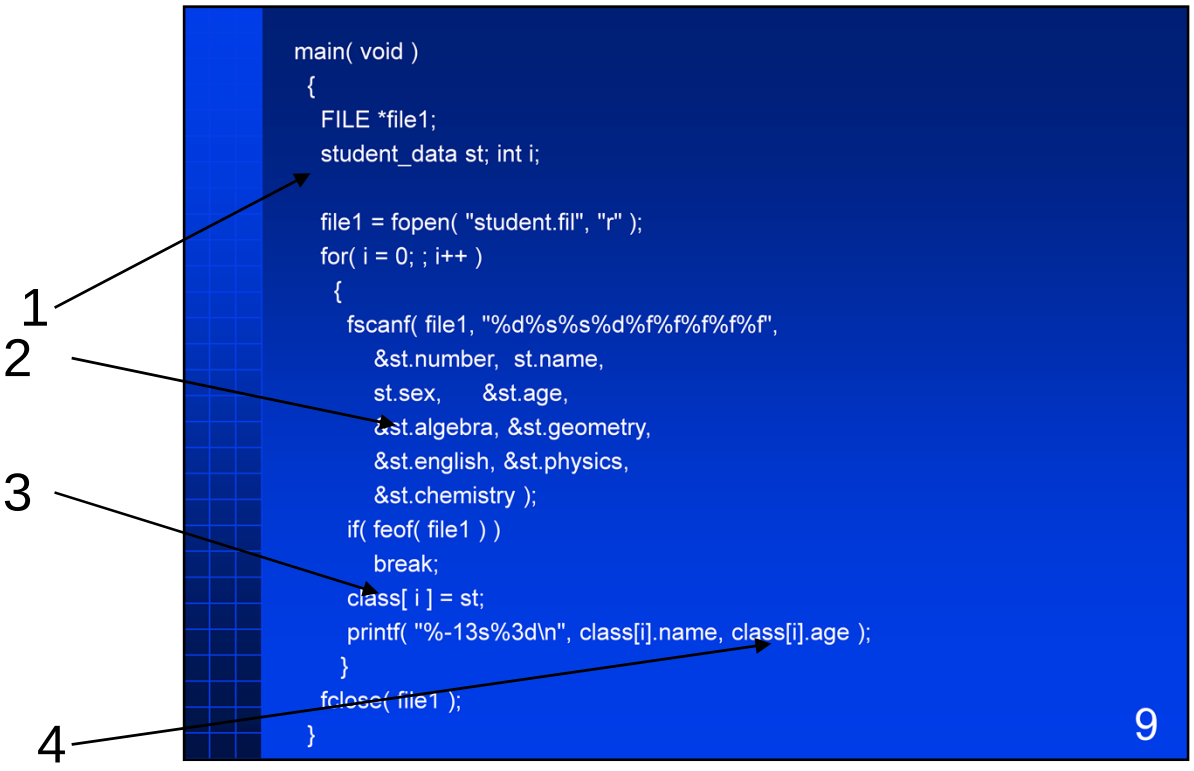
3) references to the fields. use the . operator (field selection)



Another example

1) use a type definition, similar ideas to enum. define the fields (struct student_data_fields), then define a type-name (student_data)

2) Use the type-name to define variable class as array of structs.



1) definition of local variable (single struct)

2) field references. use & as required. dot operator is higher priority that &.

3) assign a struct in a single operation

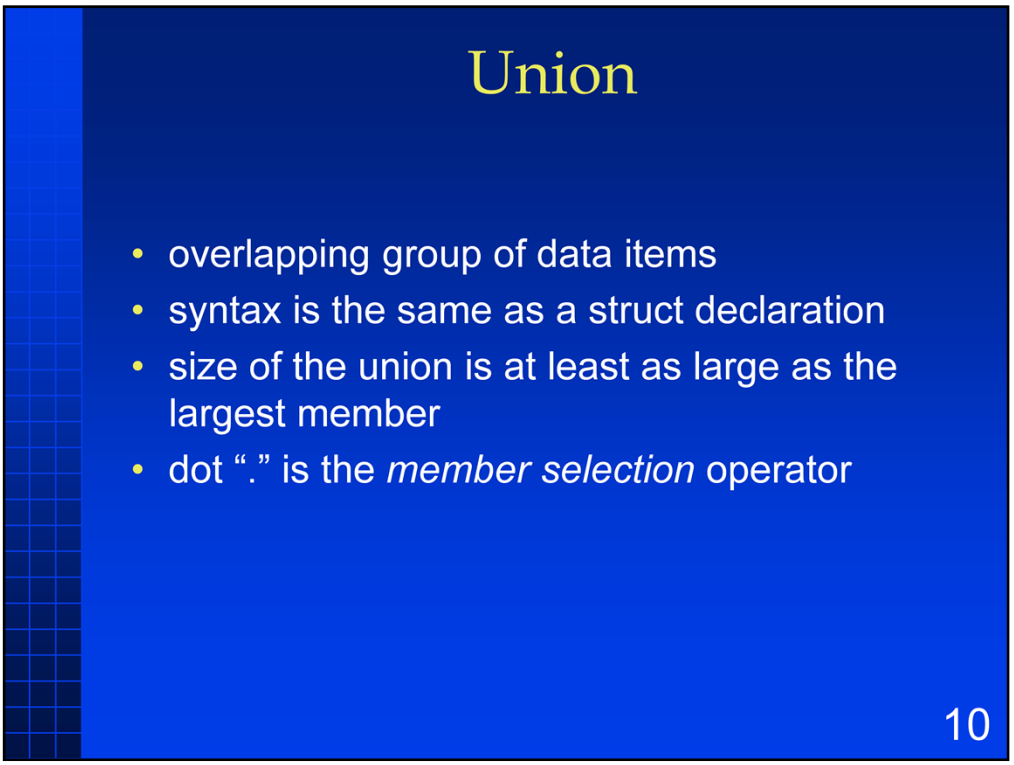
4) array subscript, then dot. choose the element from the array. it's a struct, so choose the field from the struct.

also : pointers to structs

student_data *me;

typedef student_data * sdp;

sdp me;



as stated

unions: several structs overlaid; assembler ORG, common block, redefines, variant

like struct, declares only the shape

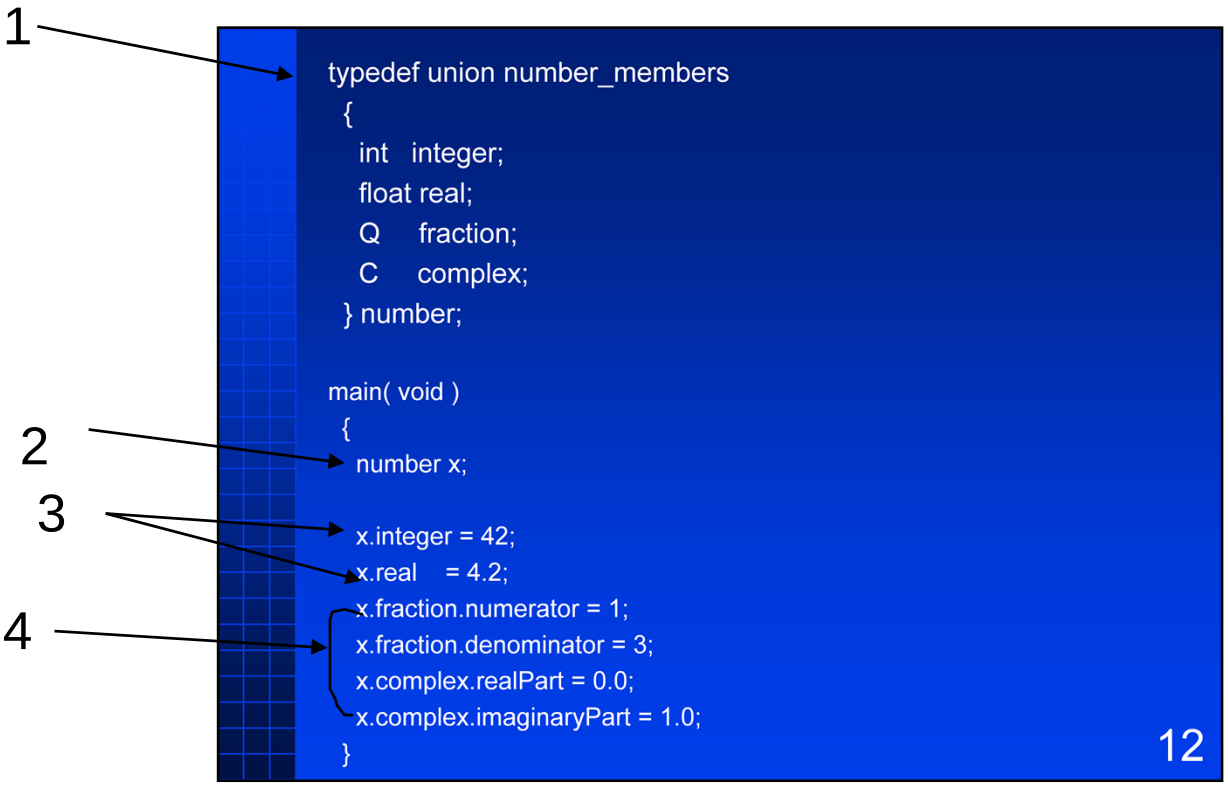
```c
/* union-1.c : example of unions */

#include <stdio.h>

typedef struct Q_fields
  {
    int numerator;
    int denominator;
  } Q;

typedef struct C_fields
  {
    float realPart;
    float imaginaryPart;
  } C;
```


define a couple of structs, to be used in next



1) declare a union, similar syntax to structs (can also be separated into two declarations, the union and the typedef)

four members:

an int called integer

a float called real

a Q called fraction -- ie a struct Q_fields

a C called complex -- ie a struct C_fields

2) define a variable

3) select the member of the union, and assign

4) select the member, then select the field