Last bit on structs, I swear

- **Structs in Debugger**

- It’s your friend
Enumerations

- new types with limited, specific values

Example: Create a *coin* type that only allows the values:

penny, nickel, dime, quarter, dollar

called symbols
Money type

enum coin {penny, nickel, dime, quarter, dollar};  // type definition
enum coin money;  // variable declaration

value money now will only accept values called penny, nickel, etc.
enums

money = dime;
money = penny;
if(money == dime) printf("dime");

penny, dime, etc are now values, like 1, ‘a’, 2.3
enums

- actual values of penny, nickel, etc are ints.
  penny == 0;
  nickel == 1;
  dime == 2; etc

- given values in order of definition

    enum coin{penny, nickel, dime...};
    
    0    1    2
values of enums

Since values are just ints, they act like ints

printf("%d %d", penny, dime);
0 2

Using enumerations let’s users not care about actual values
- Especially if the enumeration is defined in a header file
Initial values

- Enums don’t have to use default values.
  ```java
  enum coin{penny, nickel, dime, quarter = 100, half-dollar, dollar};
  
  penny = 0
  nickel = 1
  dime = 2
  quarter = 100
  half-dollar = 101
  dollar = 102
  ```
enums values

Don’t have to be unique

enum foo{fred = 1, lucy = 1, ethel = 1};
using enums

switch (money) {
    case penny: value += 1;
    break;
    case nickel: value += 5;
    break;
    case dime: value += 10;
    break;
    etc
}
one last example

char name[][12] = {
    "penny",
    "nickel",
    "dime",
    "half-dollar",
    "dollar"};

printf("%s", name[money]);

◆ only works if no symbol has been initialized.
Unions

- Memory location shared by two or more different variables

- Usually of different types, at different times.

- Generally just one at a time
Unions

- declared like structs

```c
union u_type {
    int j;
    char ch;
};

union u_type bar;
```
unions

- all variables exist in same memory
- int j & char ch share some memory.

- changing one might change the other.
access

- same as structs,

- . operator for actual union variables
- -> operator for pointers to unions
allocation

- default size is the size of the largest element.

- for dynamic allocation, use sizeof
practical example

struct symbol{
    enum type {INT, CHAR, STRING} kind;
    union foo {
        int I;
        char C;
        char S[10];
    } val;
}

mySym;
continued

    switch (mySym.kind) {
        case INT: printf("%d", kind.val.I);
            break;
        case CHAR: printf("%c", kind.val.C);
            break;
        case STRING: printf("%s", kind.val.S);
    }