**Chapter 13: Unusual Data Types**

**Structures**

* Generally want to create classes rather than structures
* Use structures to clarify data relationships
  + Bundle groups of related items together
* Use structures to simplify operations on blocks of data
* Use structures to simplify parameter lists
  + Avoid bundling data any more than is logically necessary
* Use structures to reduce maintenance

**Pointers**

* There are notes on pointers but bless python

**Global Data**

* Global variables are accessible anywhere in a program
  + Most experienced programmers have concluded that using global data is riskier than using local data
  + Most experienced programmers have concluded that access to data from several routines is useful
  + Even if they aren’t producing errors, global variables are hardly ever the best way to program

Common Problems with Global Data

* Keep information hiding and modularity at front of mind
* Major issue with Global Data
  + Inadvertent changes to global data
  + Calling the same underlying variable with two or more different names
  + Multithreading
    - Multiple threads
    - Multiple copies of the same program
  + Code reuse is hindered by global data
  + Global data complicates the picture
  + Uncertain initialization with global data
  + Modularity and intellectual manageability damaged by global data

Reasons to Use Global Data

* Preservation of global values
  + Sometimes you have data that applies to the whole program
    - State of the program
    - Data table
* Emulation of named constants
* Emulation of enumerated types
* Streamlining use of extremely common data
* Eliminating tramp data
  + Tramp data = data passed to routine or class only so it can be passed to another routine or class

Use Global Data Only as a Last Resort

* Consider the following alternatives
  + Begin by making each variable local and make variables global as needed
  + Distinguish between global and class variables
  + Use access routines

Using Access Routines Instead of Global Data

* Anything you can do with global data, you can do better with access routines
  + Access routines are a core technique for implementing abstract data types and achieving information hiding (still works even without ADTs)
* **Advantages**
  + Centralized control over the data
    - Just change source instead of everywhere its used
  + Can ensure all references to the variable are barricaded
    - Checking for overflow, out of bounds, etc
  + Get the general benefits of information hiding automatically
  + Access routines are easy to convert to an ADT
* **How to Use Access Routines**
  + Theory:
    - Hide data in a class
    - Declare that data by using “static” keywork
    - Write routines that let you look at the data and change it
    - Require code outside the class to use the access routine instead of working directly with the data
  + Guidelines for using routines to hide global data
    - Require all code to go through the access routine for data
    - Don’t just throw all global data into the same barrel
    - Use locking to control access to global variables
      * If variable is in use, any other process attempting to access it gets an error message until variable is checked back in
    - Build a level of abstraction into your access routines
      * Build at the level of the problem domain, not to implementation details

How to Reduce the Risks of Using Global Data

* In most cases, global data is really class data for a class that hasn’t been designed or implemented very well
* Develop a naming convention that makes global variables very obvious
* Create well annotated list of all global variables
* Don’t use global variables to contain intermediate results
* Don’t pretend youre not using global data by putting all data into a monster object and passing it everywhere (lmao)