Code Complete Ch. 8
Creating Data Types
Creating data types

• Much of software development has to do with modeling the concepts of a problem domain by creating new data types (i.e., abstraction)

• The native data types provided by a programming language are often too low-level to serve as effective abstractions
  – int, string, array, etc.

• By adding higher-level data types to our language, software is much easier to develop and understand
  – WebBrowser, Document, DocViewer, DocCache, etc.
Should I use a built-in type, or create a new one?

• Built-in types are capable of directly representing many kinds of data
  – int age; string name; string children[50];

• In such cases, should you create new data types?
  – Age age; Name name; List<Person> children;

• Frequently the answer is yes!
Some reasons to create new types

• Increased readability
  – "Address" instead of "string"
• Information hiding
  – char fullName[30]; // not very effective abstraction
  – Name fullName; // better
• Easier to change type definition
  – Change type from float to double
  – Important for portability
• Complex types that can't be modeled directly with built-in language types
  – Document, SpellChecker, Printer, etc.
Guidelines for creating new types

• Avoid type names that reveal internal implementation details
  – StudentHashTable  // reveals implementation details
  – StudentDatabase  // hides implementation details
Guidelines for initializing data

• Improper data initialization is one of the most fertile sources of error in computer programming

• Initialize variables as they're declared

• Declare variables close to where they're used
  – Variables don't have to be declared at the top of the method

• Check for the need to reinitialize a variable
  – Counters, accumulators, etc.

• Compiler warnings can help find uninitialized variables (-Wall)

• Check input parameters for validity