CSE 1325 Exam #1 Study Sheet (February 13, 2018)

This study sheet is provided AS IS, in the hope that the student will find it of value in preparing for the indicated exam. As always, **it is the student's responsibility** to prepare for this exam, including verification of the accuracy of information on this sheet, and to use their best judgment in defining and executing their exam preparation strategy. **Any grade appeals that rely on this sheet will be rejected.**

Definitions (understand, don't memorize):

- **Version Control -** The task of keeping a software system consisting of many versions and configurations well organized
- **Primitive type** – A data type that can typically be handled directly by the underlying hardware
- **Encapsulation** – Bundling data and code into a restricted container
- **Class** – A template encapsulating data and code that manipulates it
- **Inheritance** – Reuse and extension of fields and method implementations from another class
- **Multiple Inheritance** – A derived class inheriting class members from two or more base classes
- **Operator Overloading** – Providing a user-defined meaning to a pre-defined operator (e.g., +, ==, <<) for a user-defined type (e.g., a class)
- **Constructor -** A special class member that creates and initializes an object from the class
- **Method** – A function that manipulates data in a class
- **Instance** – An encapsulated bundle of data and code
- **Object** – An instance of a class containing a set of encapsulated data and associated methods
- **Variable** – A block of memory associated with a symbolic name that contains an object or a primitive data value
- **Operator** – A short string representing a mathematical, logical, or machine control action
- **Unified Modeling Language (UML) -** The standard visual modelling language used to describe, specify, design, and document the structure and behavior of software systems, particularly OO
- **Algorithm** – A procedure for solving a specific problem, expressed in terms of an ordered set of actions to execute
- **Getter -** A method that returns the value of a private variable
- **Setter -** A method that changes the value of a private variable
- **Software Design Pattern -** A general reusable algorithm solving a common problem
- **Singleton -** Pattern that restricts a class to instancing a single object, typically to coordinate actions across a system
- **Exception** – An object created to represent an error or other unusual occurrence and then propagated via special mechanisms until caught by special handling code
- **Declaration** – A statement that introduces a name with an associated type into a scope
- **Definition** – A declaration that (also) fully specifies the entity declared
- **Shadowing** – A variable declared in a narrower scope than that of a variable of the same name declared in a broader scope
- **Namespace** – A named scope
- **Invariant** – Code for which specified assertions are guaranteed to be true
- **Assertion -** An expression that, if false, indicates a program error
- **Data Validation -** Ensuring that a program operates on clean, correct and useful data
- **Validation Rules** – Algorithmically enforceable constraints on the correctness, meaningfulness, and security of input data
- **Intellectual Property** – Exclusive right to authors and inventors to their writing and discoveries
- **Trademark** – A symbol or name established by use as representing a company or product
- **Patent** – The exclusive right to make, use, or sell an invention, and authorize others to do the same
- **Copyright** – The exclusive right to print, publish, perform, execute, or record a creative work or its derivatives, and to authorize others to do the same

General C++ Knowledge

C++ syntax: assignments, operators, relationals, naming rules, the 4 most common primitives (bool, char, int, double) and two common classes (string, vector), instancing (invoking the constructor), for and for each, while, if / else if / else, the ? (ternary) operator, and streams (cout, cin, and getline)

#include: Difference between < > (search the system libraries) and " " (also search the local directory)

Namespaces: Purpose and how to use them. Know the membership operator ::, e.g., std::cout

Four types of scope: Global, class, local, and statement (for)

Visibility levels: private (accessible in this class and its friend functions only), protected (accessible in this class, its derived classes, and its friend functions only), and public (accessible anywhere in scope)

Difference between **declarations** (the interface, which may appear many times) and **definitions** (the implementation, which may appear only once).

The concept and value of putting declarations (interface) in .h and definitions (implementation) in .cpp, and how to write C++ code using both to minimize compile dependencies. The use of #ifndef in the header file to avoid multiple class definitions.

Difference between **normal** and **static** class members (static are shared by all derivatives of the class). Know how to define (similar to a method definition in the .cpp file, with initialization in-line)

Difference in **call by value** (normal), **call by reference** (&), and **const call by reference** (const &)

Four types of errors in programs: Compile-time, Link-time, Run-time, and Logic

Why **exceptions** are usually superior to a global error variable or returning an error code. How to create, throw, catch, and handle an exception (you may be asked to write code using exceptions!).

Difference between **cerr** and **cout**, and when and how to use each.

Concepts of **pre-conditions** and **post-conditions** for error detection in a method / function, and the use of the standard **assert** macro (from cassert) to check them. Turn off asserts by defining **NDEBUG**.

4 types of C++ classes: **enum**, **enum class**, **struct**, and **class**. Members of enum, enum class, and struct are public by default, but members of class are private by default. Struct and class are identical except for default visibility of their members.

Enum vs Enum Classes – An enum (enumeration) is just a list of aliases for integers. Enum class members are are NOT interchangeable with ints, and thus enforce type checking at compile time. An enum class can NOT include methods or other data, though.

Difference between **method**, **constructor** (a special type of class member), **delegated constructor** (also called chained constructor), and **function**. How to use **initialization lists** (sometimes called direct initialization) in a constructor.

A **default constructor** is supplied only if no non-default constructor is defined. Any number of constructors may be defined, as long as the types of each set of parameters / return types is unique.

Namespaces: Purpose and how to use them. Multiple definitions add to the namespace.

Operator Overloading: How to define an operator such as + or << (given the method / function signature, which you DON'T need to memorize). Why friend is often useful with function-based operator overloading (for access to its private variables).

Inheritance: How to derive a class from one or more base classes. Protected and public methods and field inherit. Private members, constructors, and friends to NOT inherit. How to model inheritance in UML (open-headed arrows) and how to code it in C++ (class Derived : public Base1, public Base2).

General Software Development Knowledge

Basic command line concepts such as command line flags, redirection (< and >) and pipes (|)
Basic debugger concepts such as breakpoints, watchpoints, step into, step over and how they are implemented in ddd
Basic version control concepts such as repositories ("init"), staging ("add" files), commits, and checkouts, and how they are implemented using the git command line
The basics of a simple Makefile, including rule names, dependencies, and (tab-indented) commands that will bring a rule current

Library classes, functions, and macros:

Be able to use these common C++ library members *without referencing the documentation.*
Be able to employ other library members when given documentation from cplusplus.com.

cin (from iostream) – read from STDIN *to* next whitespace, use **`cin.ignore()`** to skip next character or **`cin.ignore('\n')`** to skip *through* the next newline
getline(cin, *string_var*) (from string) – stores a line of text in *string_var*, discarding the newline
cout (from iostream) – write to STDOUT
cerr (from iostream) – write to STDERR
string (from string) **– `string s;`** to instance, **`s.append(t)`** or **`s += t`** to append t, **`s[x]`** to access char number x, **`s.size()`** to return number of chars in the vector, **`for(char c: s)`** to iterate
vector (from vector) **– `vector<type> v;`** to instance, **`v.push_back(t)`** to append t, **`v[x]`** to access element number x (just like an array), **`v.size()`** to return number of elements in the vector, **`for(auto e: v)`** to iterate through v's elements
assert (from cassert) **– `assert(Boolean)`** takes no action if true, and aborts with a message if false

UML Diagrams

Understand the use and presentation of UML extension mechanisms:

1. **Stereotype** – guillemets-enclosed **specialization**, e.g., «implements»
2. **Tag** – curly-brace-enclosed **addition**, e.g., {tag = value}
3. **Constraint** – curly-brace-enclosed **condition, restriction**, or **assertion**, e.g., {var < 16}

Understand and be able to draw the basic forms of the following UML diagrams:

1. **Top Level Diagram –** NOT UML, but often serves as a graphical index to other UML diagrams
2. **Class diagram** – Representing **class** names, **fields**, **methods**, **extensions**, and the various **relationships** between classes.
3. **Use Case diagram** – Representing **scenarios** as "action" classes (with verb-centric names) within the **system boundary** (and **actors** outside the system boundaries), relationship between actor and use case scenario, include and extend dependency, and how to read the diagram out loud ("The *actor* uses *system name* to *scenario*"). Each use case may be further documented using (1) text (2) Activity Diagram (3) Sequence Diagram (4) State Diagram (not yet covered)
4. **Activity diagram –** Displays a **sequence of activities** at the algorithm level, similar to a classic "flow chart" or "data flow diagram". Supports a single **launch point** ("dot"), arrows for **flow direction**, **branches** (decisions) with **guards** (e.g., [valid_order]), multiple execution paths (**fork** and **join**), **hierarchical** activity bubbles to an arbitrary depth, **interrupts** within **regions**, object definition as data payloads, **swim lanes** for aggregating a subset of activities (similar to tags, but assigned visually), and multiple **termination points** ("targets").

Software Design Patterns

Patterns are discovered, not invented, and validated by the community at large (meritocracy). You should be able to match the definition to the pattern, and *recognize* the pattern in UML or code, but you will NOT be asked to write a C++ class or draw the UML for a pattern on the exam.

Singleton – Restricts a class to instancing a single object, typically to coordinate actions across a system. The C++ implementation makes the constructor private, and provides a public get_instance method in its place that returns the one private static instance of the Singleton class.

Decorator - Dynamically adds new functionality to an object without altering its structure. This is different from inheritance, which statically adds functionality. Decorators can be nested as deeply as desired.

Model-View-Controller (MVC) – Separates the business logic (the Model) from input (the Controller) and output (the View). This is particularly useful for providing multiple I/O options, e.g., a command line interface, a graphical user interface, and an automated test interface.

Intellectual Property

Know the three primary types of intellectual property in the USA of primary interest to computer science and engineering professionals:

- **Trademark –** Assigned on first use of a brand (if unique) but may be registered, and is retained until abandoned. Intended to avoid confusion of brands in the marketplace.
- **Patent** – Assigned for an invention (including new computer hardware and software technologies) only after application and review, and is usually valid for 14 years. Gives the inventor the right to control the use of their invention.
- **Copyright –** Assigned when creating a work of authorship (software source code as well as literary, artistic, dramatic, etc.), but may be registered, and is retained essentially for a lifetime. Gives the author exclusive control over copying, performing, and modifying their work.

Also know the primary types of software licenses, generally from least to most restrictive:

- **Public Domain** - all ownership is disclaimed (SQLite)
- **Permissive** (MIT, BSD, Apache) permits use, copying, distribution, and (usually with attribution) derivatives, even proprietary derivatives (BSD Unix, Apache)
- **Protective** (GPL 2, 3, Lesser GPL, EPL) permits use, copying, distribution, and derivatives with share-alike rules (Linux, Gimp). GPL 3 also includes important patent clauses.
- **Shareware** permits (sometimes limited) use, copying, and (usually) distribution (Irfanview, early WinZip releases)
- **Proprietary** permits (often restricted) use (Windows, Photoshop)
- **Trade Secret** typically restricts use to the copyright holder

Also Study and Know the Quick Reviews for Each Lecture!