CS2124 Final Exam Coverage - 25S

Basics

Lec 1-3; rec01,2; hw1,2

- Types
    - static typing
    - Common types: primitive types, strings and vectors
    - Default values for non-primitives
- Conditions / if / else
- Looping: for loop, while, do-while, ranged for. Also break and continue.
- Console I/O
- File input: open and closing files. Testing if open works. Reading files.
- Functions / parameter passing / return types. Default parameter values
    - Use of by-value, by-reference and by-constant-reference with ranged for
- Defining types: structs
    - Filling a vector with instances of a struct.

OOP Basics

Lec 4,5 / rec03 / hw3

- Encapsulation and data hiding
    - meaning of public / private.
- Constructors
    - Initialization lists
        - order of member variable initialization
    - Default constructor
    - Used with emplace_back
- Methods
    - const methods
    - getters / setters
- Nested classes
- Delegation
- Overloading output operator
- Difference between the keywords `struct` and `class`

Pointers and Dynamic Memory

Lec 6,7 / rec04,5 / hw4,5

- Association
- Addresses
- address-of operator: &
- Pointer variables and strict typing
- dereference operator: `*`
- `nullptr`
- `this`
- arrow operator: `->`
- operator precedence: dot vs. asterisk
- Issues: dangling pointers and memory leaks
- **`new`** / **`delete`**
- pointers and `const`

Copy Control and the Vector class implementation

Lec 8-11; rec05,6;

- Copy control:
  - destructor,
  - copy constructor,
  - assignment operator
- dynamic arrays: **`new[], delete[]`**
- pointer arithmetic: **`p[k] == *(p+k)`** for all integer values of k and pointers p.
- Index operator: **`operator[]`**
  - Overloading methods based on const
- keyword: **`explicit`** and implicit conversions
- ranged for support, i.e. **`begin()`** and **`end()`** methods.

Operator Overloading

Lec 5, 8-12; rec07;

- Implement as member or non-member
- Restrictions on operator overloading
  - Arity, associativity, precedence
  - Operators must already exist for primitives
  - Operators cannot be overloaded if only arguments are primitives
- Some of the operators that we have overloaded:
  - `<<, >>, =, +=, ==, <, <=, >, >=. != +, ++, --, []`
- Conversion operators, e.g. **`operator bool`**

Cyclic Association and Separate Compilation

Lec 13, 14; rec08; hw06

- Forward class declaration
- Header and implementation files
- Include guards
- Namespaces

Inheritance

Lec 15-20; rec09,10; hw08.

- Terminology:
    - Base / Derived, Parent / Child, Ancestor / Descendant, Super / Sub class.
- What you inherit and what you don't
- Principle of substitutability
- Slicing
- Polymorphism. What does it mean? What is required?
- Initialization and derived classes
- Inheritance and pointers
- Overriding vs overloading
- Calling base class methods from a derived class method
- Keywords: `override`, `final`, `protected`
- Abstract methods and classes
- Method hiding
- Copy control and inheritance
- Calls to virtual methods from inside constructors
- Multiple inheritance

Linked List

Lec 20-22; rec11,12; hw09

- Singly linked lists defined as a `Node*`
- const and pointers! (yes, again)
- Doubly linked lists
- Parameter passing and lists
- Defining a doubly linked list class
    - sentinels
    - iterators

Implementing Iterators

Lec 22, 23; rec12

- Implementing an iterator publicly nested class for our Vector class
- Implementing an iterator publicly nested class for our List class
- const and iterators, const_iterator

Templates, the STL, …

Lec 24-25; rec 13

Implementing class and function templates

- STL collections
- STL algorithms
    - **half open range**
    - `sort`, `find`, `find_if`
        - predicates
- ~~functors and~~ lambda expressions and lambda capture
- `auto` (but **don't use on the exam** unless the question says you can)
- `pair`, `map`, `set`

Recursion

Lec 25-27; rec14

- How does it work?!?!
- Design; of a recursive function
    - how will the recursion be used?
    - Once you have that, what else do you need to do?
        - Before? After?
    - What can be passed that doesn't require recursion, aka *base case*
- Examples:
    - duplicate a list, towers of hanoi, print digits, print / count bits, tree sum, fibonacci
- Reading recursive functions
- Impact on call stack

Exceptions and Assertions

Lec 27, 28; rec 14

- `try` / `catch`
  - ordering of catch clauses
  - `...`
- `throw`
  - what can be thrown
  - re-throwing
- exception hierarchy
  - `exception, out_of_range`
  - `what()`
  - Use of polymorphism
- `assert`
  - `NDEBUG`

Not Covered

- Multi-threading
- Smart Pointers, e.g. unique_ptr