David Liang

Lecture # 4 pil 10/4/16

- inline function
  - precede a function definition and causes the compiler to remember and eliminate overhead.
  - it speeds up the process since it calls overhead variables on stack ~~and~~ but it enlarges the target.
  - Better type-safe, and better scope management.

  ex. static inline int inc(int* x) {return ++(*x);}
  - Has to use -O3 to make sure inline is ~~a~~ implied.

- Variadic Function

  ex. $\text{int avg(int count, } \underbrace{\text{...}}\text{)}$

  ↳ means unknown amount of parameter
  - acts like a function that takes unknown # of argument.

- Pointer to Function
  - pointer has the ability to change a variable by reference or address/value of.
  - '->' means to dereference and call by value at the same time.
  - using '*' tells the compiler to look at the source and manage it from there than to mess with a temporary value.

- Overloading operator
  - Based on the ex.
    - you can overload an operator based on class reference.
    - or overload the entire operator of a library for the whole program.

- Trailing Return Types.

  ex. auto function_name (<parameters>) -> trailing return type.
  - Based on parameters.
  - It utilizes the decltype specifier
  - Lambda return type

- Namespace.
  - If I have multiple different namspaces with the same name for either struct or classes use:

    ex. circle::geometric_object. -> It uses circle's geometric object

- think about enum / classes if you will.
  Does namespace act like a class in a way?

- Run time
  - usually when the compiler figures things out like auto return.

- General Constant Expression
  - constexpr
    ex. constexpr circle(int r): radius(r){}
    - this tells the compiler to do it (calculation) during compile time.
    - It can not be used with lvalue reference (error) has to be a Rvalue.

- Lambda Function
  - [capture] (params) -> return type {body}
    ^specifies what you what back. ^arguments.

- -error exception
  - try{ }
    catch{ ——— what to throw in case / what do you expect is the error}

- ~~Leet~~

David Liang

Lecture #5 10/6/16

- Object Model
  - Abstract, Encapsulation, Modularity, Hierarchy
  - minor element:
    Typing, Concurrency, persistence.

- Abstraction
  - to cope with complexity
  - arise from similarities and simplifies description of specifications.
  - Abstraction concept qualifies if it can be described, understood, and analyzed
  - Denote essential characteristics of an object
    - defined conceptual boundaries (h files)
  - Focuses on outside views and avoids surprises.

- -Types
  - Entity - a useful model of a problem domain.
  - Action - generalized set of operations.
  - Virtual Machine - operations used by some superior level of control.
  - Coincidental - package a set operations that has no relation to one another.

- Encapsulation
  - Implementation of private abstraction.
    - provides explicit barriers
  - Forming the structure and behavior of an abstraction
    - separate the interface and its implementation

- Modularity
  - Separating a program into components.
  - Creates well-defined boundaries - classes, and objects (c++)
  - Programs (separated) can comple separately as well

- Hierarchy
  - a set of abstractions offers a hierarchy
    - ranking of absractions :,
    - -class and object are important structure

- Typing
  - Precise characterization of properties which all entities all share.
  - It is enforcement of the class of an object
    - the idea of conformance.
  - type consistency and time of type binding
  - Polymorphism - strongly and statically typed.
    - very powerful next to absraction

- Concurrency
  - Handles many dif events simultaneausly
    - Computations pass single processor.
  - it focuses on abstraction and synchronization.

- Persistence.
  - Property of an object which exceeds time /or space.
  - It is
    - results in exp. evaluation
    - Local variable -> initialization
    - Between executions
    - that outlives a program

- Class
  - A template that group operations and related data together.
  - Provides
    - Generalization , Abstraction
    - Scoping , Hierachy
  - ~~#~~ Used as reusability

- Object
  - Instatiation of one or more class.
  - Has
    1) state - data members
    2) Behavior - separates from objects of the same class
    3) Identity - represented by method.

- Classes in C++
  - starts with 'class'
  - operates on data member
    - constructors - create and initalize
    - Destructors - frees the state
    - modifiers - alter the state
    - selectors - access the state
    - iterators - permits parts to be accessed in order.
  - Data members and methods - defined order restriction
    - Levels:
      1) public
      2) Protected - accessed by all of the ~~[illegible]~~ derived classes
      3) private - only to class itself
      4) friend - allows a class to access everything (no...).