New Mexico C++ Programmers

Intro to C++ Classes

LOUIS LANGHOLTZ 03/16/2023

Classes?

- Of things (shape, person, location, etc.)
- And instances of classes.
- Compound types.
- More generally, user defined types.
- Object oriented programming: *is-a*, *has-a*, *encapsulation, inheritance*, *polymorphism.*

Use categories

- Aggregation.
- Encapsulation.
    1. Implementation.
    2. Invariance.
- Dominion.
- Other?

Brought to you by C

- From C: `struct`. Compound types with member variables.
- C++ drops: need to preface usage with `struct` (usually).
- C++ adds: `class`, *access specifiers*, member functions, inheritance, `virtual`.
- Difference between `struct` and `class` is their default access:
  - `struct`: members are public - accessible where defined.
  - `class`: members are private - accessible only within member functions (or “friends”).
- Otherwise interchangeable - except… some compilers don’t want to see a class referred to as both (MSVC).
- *Access specifiers*:
  1. `public`.
  2. `private`.
  3. `protected` - private except to classes that have *inherited* from this class.

Member Functions

- *Static*: non-instance accessing functions.
- *Non-static*: instance accessing functions:
  - Non-“special” member functions.
  - “Special” member functions. Functions that may be defined by compiler even if not defined by user!

“Special” Member Functions

- Specific functions **automatically** defined by compiler, unless prevented by programmer:
    1. Default constructor.
    2. Copy constructor.
    3. Move constructor.
    4. Copy assignment.
    5. Move assignment.
    6. Destructor.
- A framework for *regularity*, reducing surprise!

Inheritance

- Uses the access specifiers also but meaning is a little different:
  1. `public` (default for `struct`): For “is-a”. But, be wary of *squares & rectangles & change!*
  2. `private` (default for `class`): For “is implemented in terms of”. But, prefer composition.
  3. `protected`: private except to classes that have *inherited* from this class.
- Examples:
  - `struct base { int b; int f(); }; base base_object;`
  - `struct sd: base { int v; int g(); }; sd a_base_object;`
  - `class cd: base { int v; int g(); }; cd an_implemented_object;`

Invariance

- English language definition: “The property of remaining unchanged regardless of changes in the conditions of measurement”.
- A property of variables or relationships between them.
- Enforced by hiding class definition, non-public access specifiers, or constructors throwing exceptions.
- A cause for *encapsulation.*

Virtual

- Base classes. Help with multiple inheritance.
- Member functions.
  - A way to provide type system based, dynamic polymorphism.
  - Be aware of alternatives: static polymorphism, NVI, strategy pattern, traits.
  - Require user defined virtual destructor to ensure proper destruction of derived types.
  - Cause issues with copy operations and complicate “equality”.
  - Leads aware from *value semantics* to *reference semantics*.

Value semantics v. Reference semantics

- Value semantics == local reasoning. Passing by value or by constant reference. Easier to reason about.
- Reference semantics == remote reasoning. Passing by non-constant reference. Having pointers or references to variables that some other code may change. Harder to reason about.

Preferences & opinions

- C++ Core Guidelines C.2: “Use class if the class has an invariant; use struct if the data members can vary independently”.
- C++ Core Guidelines C.10: “Prefer concrete types over class hierarchies”.
- Me: Struct first design. Use struct, public member variables, & free functions.
- When you decidedly have an invariant, make the member variables private and provide public member functions that can’t violate the invariant.
- Avoid privatization.
- Avoid derivation.
- More than two member variables? Have multiple invariants or no invariants! Single responsibility principal: refactor the member variables with invariant properties or relationships to their own classes.

More Examples

1. `struct point {int x; int y;};`
2. `class point {int x; int y;};`
3. `class point {int x; int y; public: int get_x() const {return x;} void set_x(int v) {x = v;}};`
4. `class point {int x; int y; public: point(int x_, int y_); int get_x() const; void set_x(int v);};`
5. `struct point {int x; int y; public: int get_x() const {return x;} void set_x(int v) {x = v;}};`

Questions & Answers

- Hope this material has given you some introduction to C++ classes!
- Did it provide any “food-for-thought”?
- Have you heard the term “encapsulation” before?
- What’s more encapsulated? Example 1 or example 4?
- Are you familiar with object oriented programming?
- What about “regularity” and polymorphic base classes?