An Introduction to Object-Oriented Programming Using C++

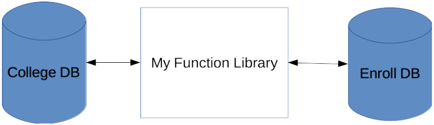
Eden Burton <ronald.burton@senecacollege.ca>

16/09/2015

Complexity

- IPC144, using algorithms to build functions
- OOP244, organizing functions (or "methods") and related data into larger systems

Complexity - A Procedural Approach



College DB

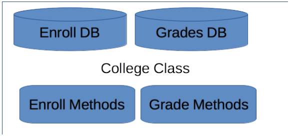
My Function Library

Enroll DB

```
int enrollStudent(int studentNo,
   struct EnrollDB* s, int course)  {
      ...
}
void enterAvgGrade(int studentNo,
   struct GradesDB* c, int grade)  {
      ...
}
....
```

– data and activities are logically separated

Complexity - An Object-Oriented Approach



```
class College {
   struct EnrollDB* s;
   struct GradesDB* g;

   int enrollStudent(int studentNo, int course){
   }
   void enterAvgGrade(int studentNo, int grade){
   }
}
```

- data and their related activities are combined into classes

Types

- 2 "types" of types
  - fundamental (int, char)
  - user defined (structs, classes)
- allows developer to declare what data should be stored in variables
- complier allocates a fixed amount of memory
- hardware has no notion of type
- declared types help compiler find problems

Type Example

Namespaces

- – creates a scope in which variables are defined
- – :: allows you to explicitly refer to a space
- – `using` allows you define a default spece
- – std is the standard namespace for C++

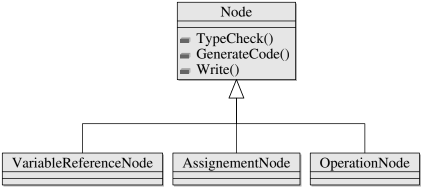
Namespace Example

Abstraction

- identify high level parts of a solution…
  - ignore implementation details
  - focus on identifying relationships between parts
- defining interfaces is a means of abstraction

UML

- visual way of describing abstraction
- class diagrams show properties and relationships between them
  - class name
  - attributes (data members)
  - operations (member functions)



Classes vs Objects

- a user-defined type
- similar to `structs` but also contain operations
- are used as "blueprints" to create objects

```
class Box {
      double length;
      double breadth;
      double height;
      int getVolume(){...};
};

int main ()  {
   Box b;
   cout << "Volume=␣" << b.getVolume() endl;
}
```

Object-Oriented Development

"An object has state, behavior and identity ..."
Booch

- state, internal data, attributes
- behaviour, via method definition
- identity, pointer to memory

Key Principles

- encapsulation
- polymorphism
- inheritance

Encapsulation

Binding data with the code that manipulates it

- make fields of a class private
- provide client access to the fields via declared interfaces
- keeps data safe from external interference

Class-Based Inheritance

Allows classes to be defined in terms of other classes.

- – reuse mechanism
- – classification

```
class Shape  {
   public:
      void setWidth(int w) { width = w; }
      void setHeight(int h) { height = h; }
   protected:
      int width; int height; };
class Rectangle: public Shape {
   public:
      int getArea() { return (width * height); }};
int main(void) {
   Rectangle Rect;
   Rect.setWidth(5); Rect.setHeight(7);}
```

Polymorphism

Behaviour changes depending on context

- a class can be used in place of its base class
- prevents implementation dependent code

Polymorphism Example

Modules

- a division of an application
- ideally a highly cohesive part of code
  - should be implementable and testable independently
- compiler builds a binary unit that can be linked with other ones

C++ module should consist of

- header file (.hpp) which contains the interface clients use to access it
- implementation file (.cpp) which contains the actual implementation logic

Module Example