Scientific Computing: Objects

Erik Spence

SciNet HPC Consortium

16 January 2014

SciNet compute • calcul CANADA

Today's class

Today we will discuss the following topics:

- Object-oriented programming.
- Classes and objects in C++.
- Class inheritance, functional polymorphism.
- Assignment 2.

SciNet compute • calcul CANADA

Limits to structured programming

Structured programming is the methodology of breaking up a given programming task into smaller bits using subroutines which access other subroutines, until one has simple tasks to implement. This is a good approach to solving one particular programming task.

Problems arise with this approach when there are multiple different tasks being applied to the same data:

- Each separate code needs to know about the data structure.
- Leads to reinventing the wheel.

In these situations one would instead like to build 'components' with known properties and known interfaces to allow them to easily plug into your program.

SciNet compute • calcul CANADA

Object-oriented programming

What is object-oriented programming (OOP)?

- Structured programming: functions and data are accessible from everywhere and by everyone.
- OOP: functions and data are bundled together into a 'class'. The implementation details are hidden.

OOP has a number of advantages:

- Complexity can be hidden inside each class.
- Separates the interface from the implementation.
- Good reuse of components.

But be sure you know:

- the computational cost of the operations.
- what temporary variables are created.

At a low level, OOP may need to be broken for best performance.

SciNet compute • calcul CANADA

C++ class syntax

How is OOP accomplished in C++? Classes and objects!

- Define a class, with its own internal variables and functions.
- Declare variables (objects) to be of this class.

In this sense a class is like a customized variable, a self-contained package that includes all the bits that you need to work on your data.

```
class classname {
 private:   // Things that can't be accessed from outside the object.
   int var1;
   float var2;

 public:   // The things that can be accessed from outside.
   void func1();
   int func2(int x, int y);
};
```

SciNet compute • calcul CANADA

Data hiding

- Good implementations hide the data details.
- Each function or variable is one of three types:
  1. private: only functions of the class have access.
  2. public: accessible when using the class.
  3. protected: accessible only to this class and derived classes.
- These are specified as sections within the class.

```
class classname {
  // Things that can't be accessed from
  // outside the object.
  private:
    int var1;
    float var2;

  // The things that can be accessed from
  // outside the object.
  public:
    void func1();
    int func2(int x, int y);
    float ReturnVar2() {
      return var2;
    };

};
```

SciNet compute • calcul CANADA

The StoneWt class

```
// MyStoneWt.cpp
#include <iostream>
class StoneWt {
  private:
    int stone; // whole stones
    float lbs_left; // fractional pounds
  public:
    void set_stn(int stn, float lbs) {stone = stn; lbs_left = lbs;};
    void show_stn() {std::cout << "The weight is " << stone
      << " stone and " << lbs_left << " pounds." << std::endl;};
};
```

The StoneWt class

```
// MyStoneWt.cpp
#include <iostream>
class StoneWt {
  private:
    int stone; // whole stones
    float lbs_left; // fractional pounds
  public:
    void set_stn(int stn, float lbs) {stone = stn; lbs_left = lbs;};
    void show_stn() {std::cout << "The weight is " << stone
      << " stone and " << lbs_left << " pounds." << std::endl;};
};

int main() {
  StoneWt myweight;
  myweight.set_stn(10, 2.1);
  myweight.show_stn();}
```

The StoneWt class

```
// MyStoneWt.cpp
#include <iostream>
class StoneWt {
  private:
    int stone; // whole stones
    float lbs_left; // fractional pounds
  public:
    void set_stn(int stn, float lbs) {stone = stn; lbs_left = lbs;};
    void show_stn() {std::cout << "The weight is " << stone
      << " stone and " << lbs_left << " pounds." << std::endl;};
};

int main() {
  StoneWt myweight;
  myweight.set_stn(10, 2.1);
  myweight.show_stn();}
```

```
ejspence@mycomp ~> g++ MyStoneWt.cpp -o MyStoneWt
ejspence@mycomp ~> ./MyStoneWt
The weight is 10 stone and 2.1 pounds.
```

The StoneWt class, improved

```
// StoneWt.h
#ifndef STONEWT_H
#define STONEWT_H

class StoneWt {

  private:
    int stone; // whole stones
    float lbs_left; // frac. lbs

  public:
    // Set the weight, in stone.
    void set_stn(int stn,
                 float lbs);

    // Print the weight.
    void show_stn();
};
#endif
```

Stolen from C++ Primer Plus

```
// StoneWt.cpp
#include <iostream>
#include "StoneWt.h"
void StoneWt::set_stn(int stn, float lbs) {
  stone = stn; lbs_left = lbs;
};

void StoneWt::show_stn() {
  std::cout << "The weight is " <<
    stone << " stone and " << lbs_left << "
    pounds." << std::endl;
};
```

```
// MyStoneWt.cpp
#include "StoneWt.h"
int main {
  StoneWt myweight;
  myweight.set_stn(10, 2.1);
  myweight.show_stn();
}
```

Constructors

C++ allows the flexibility of instantiating your object in different ways. This is accomplished through a 'contructor':

- Constructors are functions that have the same name as the class.
- They perform whatever initialization of the object is necessary.
- By having different numbers or types of arguments you can have different constructors to do different initializations.
- The constructors are listed in the public part of the class definition.

```
class classname {
  ...
  public:
    // Two constructors.  Note that the constructors do not have return types.
    classname();       // Default contructor.
    classname(arguments);
    int func1(int a);
};
```

SciNet compute • calcul CANADA

Destructors

By default C++ will destroy a variable when it 'goes out of scope'. But some objects require special instructions to be deleted correctly. This function is called the 'destructor'.

- The destructor is a function that has the same name as the class, but begins with a ~.
- The destructor must perform the deletion of whatever variables were created with 'new'. Otherwise it needn't be specified.
- The destructor is listed in the public part of the class definition.

```
class classname {
  ...
  public:
    // Note that the destructor does not have a return type.
    ~classname();
};
```

SciNet compute • calcul CANADA

The StoneWt class

```cpp
// StoneWt.h
#ifndef STONEWT_H
#define STONEWT_H

class StoneWt {

  private:
    int stone; // whole stones
    float lbs_left; // frac. lbs

  public:
    StoneWt();
    StoneWt(int stn, float lbs);
    ~StoneWt();
    void set_stn(int stn,
                 float lbs);
    void show_stn();
};
#endif
```

```cpp
// StoneWt.cpp
#include <iostream>
#include "StoneWt.h"

void StoneWt::set_stn(int stn, float lbs)
{stone = stn; lbs_left = lbs;};

void StoneWt::show_stn() {
  std::cout << "The weight is " <<
    stone << " stone and " << lbs_left <<
    " pounds." << std::endl;
};

StoneWt::StoneWt() // Default constructor.
{stone = 0; lbs_left = 0.0;};

StoneWt::StoneWt(int stn, float lbs)
{stone = stn; lbs_left = lbs;};

StoneWt::~StoneWt() {}; // Destructor.
```

The StoneWt class, continued

```
// MyStoneWt.cpp
#include "StoneWt.h"
int main {
  StoneWt myweight;
  StoneWt *myweight2;

  myweight2 = new StoneWt(4, 0.2);
  myweight.show_stn();
  myweight.set_stn(10, 2.1);
  myweight.show_stn();
  myweight2 -> show_stn();
}
```

The StoneWt class, continued

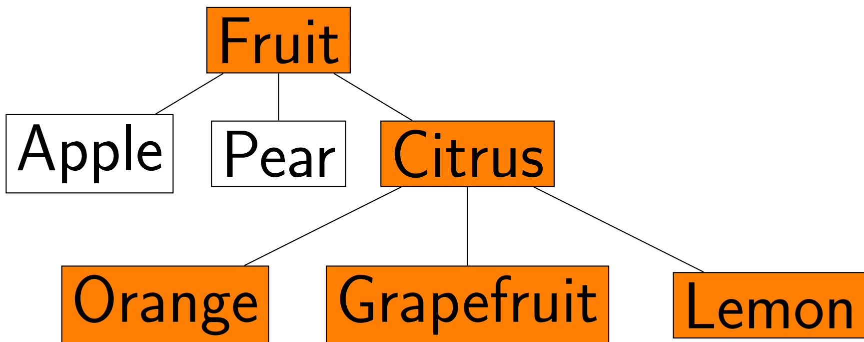
```
// MyStoneWt.cpp
#include "StoneWt.h"
int main {
  StoneWt myweight;
  StoneWt *myweight2;

  myweight2 = new StoneWt(4, 0.2);
  myweight.show_stn();
  myweight.set_stn(10, 2.1);
  myweight.show_stn();
  myweight2 -> show_stn();
}
```

```
ejspence@mycomp ~> g++ StoneWt.cpp -c -o StoneWt.o
ejspence@mycomp ~> g++ MyStoneWt.cpp -c -o MyStoneWt.o
ejspence@mycomp ~> g++ StoneWt.o MyStoneWt.o -o MyStoneWt
ejspence@mycomp ~> ./MyStoneWt
The weight is 0 stone and 0 pounds.
The weight is 10 stone and 2.1 pounds.
The weight is 4 stone and 0.2 pounds.
```

Class inheritance

Sometimes it makes sense to build classes that are based on the features of other classes (modular programming!), rather than starting a new class from scratch.



Class inheritance, continued

- Derived classes are derived from base classes.
- The derived classes automatically include the base class' public members.
- When the derived-class constructor is invoked, the base-class constructor is also invoked. Hence the option of specifying how the base-class constructor is to be invoked by the derived-class constructor.

```
// Base class
class baseclass {
  private:
    ...
  public:
    baseclass();
    baseclass(int a);
    ...
};
```

```
// Derived class
class derivedclass : public baseclass {
  private:
    ...
  public:
    derivedclass() : baseclass();
    derivedclass(int a) : baseclass(a);
    ...
};
```

compute • calcul CANADA

Class inheritance, example

```
// matrix.h
#ifndef MATRIX_H
#define MATRIX_H

class matrix {
  // 'protected' allows these
  // variables to be directly
  // accessed by derived classes.
  protected:
    int rows, cols;
    double *elements;
  public:
    matrix(int r, int c);
    ~matrix();
    int get_rows();
    int get_cols();
    void fill(double value);
};
#endif
```

```
// sqrmatrix.h
#ifndef SQRMATRIX_H
#define SQRMATRIX_H
#include <iostream>
#include "matrix.h"
class sqrmatrix : public matrix {
  public:
    sqrmatrix(int r,int c) : matrix(r,c) {
      if (r != c) {
        std::cerr<<"Not a square matrix.";
        exit(1);
      }};
    double trace() {
      double sum = 0.0;
      for(int i = 0; i < rows; i++)
        sum += elements[i * cols + i];
      return sum;
    };
};
#endif
```

Class inheritance, example

```
// Mysqrmatrix.cpp
#include <iostream>
#include "sqrmatrix.h"
int main {
  sqrmatrix Q(5,5);

  Q.fill(1.6); // Assume that this fills the matrix with elements.
  std::cout << "Trace = " << Q.trace() << std::endl;
}
```

```
ejspence@mycomp ~> g++ matrix.cpp -c -o matrix.o
ejspence@mycomp ~> g++ Mysqrmatrix.cpp -c -o Mysqrmatrix.o
ejspence@mycomp ~> g++ Mysqrmatrix.o matrix.o -o Mysqrmatrix
ejspence@mycomp ~> ./Mysqrmatrix
Trace = 8
```

Polymorphism

Polymorphism refers to the use of a standard set of properties and behaviours so that objects can be used interchangably. This is implemented by the overloading and overriding of previously-existing functionality.

Why bother?

- Avoid the duplication of code.
- Re-using function names for the same functionality allows a common interface, and consistency of design.
- This simplifies and structures the code.

SciNet compute • calcul CANADA

Polymorphism in class inheritance

Polymorphism naturally arises in class inheritance, since the derived class shares commonality with the base class. The idea is as follows:

- Use the base class as a framework for the derived class' usage.
- Override base class functions with new implementations of the functions in the derived class definitions.
- Simplifies and structures the code.

However, there is a twist, because baseclass pointers can point to derivedclass objects (think about it):

- However, which overloaded function will it run, the baseclass function or the derivedclass function?
- If you want it to run the derivedclass function, define the baseclass version of the function with the virtual keyword.

SciNet compute • calcul CANADA

Poymorphic class inheritance, example

```
// matrix.h
#ifndef MATRIX_H
#define MATRIX_H
#include <iostream>
class matrix {
  protected:
    int rows, cols;
    double *elements;
  public:
    matrix(int r, int c);
    ~matrix();
    int get_rows();
    int get_cols();
    virtual void printstats() {
      std::cout << "rows: " <<
      rows << ", cols: " << cols
      << std::endl;
    }
};
#endif
```

```
// sqrmatrix.h
#ifndef SQRMATRIX_H
#define SQRMATRIX_H

#include "matrix.h"
class sqrmatrix : public matrix {
  public:
    sqrmatrix(int r,int c) : matrix(r,c) {
      if (r != c) {
        std::cerr<<"Not a square matrix.";
        exit(1);
      }};
    double trace();
    void printstats() {
      std::cout << "rows: " << rows <<
      ", cols: " << cols <<
      ", trace: " << trace() << std::endl;
    }
};
#endif
```

Class inheritance, example

```
// Mysqrmatrix.cpp
#include "sqrmatrix.h"

int main {
  matrix Q(2,3);
  sqrmatrix R(5,5);
  matrix *S = new sqrmatrix(6,6);

  Q.fill(1.6); R.fill(1.6); S -> fill(1.6);
  Q.printstats(); R.printstats(); S -> printstats();
}
```

```
ejspence@mycomp ~> make
ejspence@mycomp ~> ./Mysqrmatrix
rows: 2, cols: 3
rows: 5, cols: 5, trace: 8
rows: 6, cols: 6, trace: 9.6
```

If the matrix class printstats function was not virtual then "S -> printstats()" would print the matrix class version.

Assignment 2

On the class website is a file called Diffuse.cpp. This program evolves a 1D density field, and outputs the results to a text file. Assignment: put this code under version control with git and modify it into a new, modularized version:

- Create a class for the field being evolved that contains the functionality to allocate the field, perform a single time step, constructor, destructor, etc.
- Create a Makefile to compile this project.
- Create a module containing the constants for the problem.
- Create a module for the file output functions.

The orginal file should become just a 'driver', containing main, that solves the same problem as the original code. Please submit:

- all source, header and make files for the new program.
- the output of 'git log' for your code development. We expect to see several commits, and meaningful comments.

SciNet compute • calcul CANADA