PY410 / 505
Computational Physics 1

Salvatore Rappoccio

Code

- Code is in CompPhys/ReviewCpp/ClassExamples

Classes

C++: Classes

- You can define your own data types in C++
- These are called “classes”
- They are an aggregate of information:
  - Data members:
    - data for the class
  - Methods:
    - functions to operate on the class

- Example: member data, no methods:

```
class Point {
 public:
  double x;
  double y;
};
```

C++: Classes

- Access member data in two way:

- if a value: dot (a.value)
- if a pointer: arrow (b->value)

```
  Point p1;
  Point p2;
  p1.x = 0.;
  p1.y = 1.;
  p2.x = 2.;
  p2.y = 3.;
  std::cout << "p1: (" << p1.x << "," << p1.y << ")" << std::endl;
  std::cout << "p2: (" << p2.x << "," << p2.y << ")" << std::endl;
  Point * p = &p1;
  std::cout << "p : (" << p->x << "," << p->y << ")" << std::endl;
```

C++: Classes

- Methods: functions defined WITHIN a class:

```
class Point {
 public:
  double x;
  double y;

  void print() const {
    std::cout << "(" << x << "," << y << ")" << std::endl;
  };
};
```

Have access to the data members for “THIS” object!

- These are only accessible when you have an OBJECT of or a POINTER to the class:

```
  std::cout << "p1: ";
  p1.print();
  std::cout << "p2: ";
  p2.print();
  std::cout << "p : ";
  p->print();
```

Cannot call “print()” without an object!

C++: Classes

- Within a class, you can use a special pointer called “this”
- It is a pointer to “this” class
- Thus, these are equivalent:

```
  void print() const {
    std::cout << "(" << x << "," << y << ")" << std::endl;
  };
```

```
  void print() const {
    std::cout << "(" << this->x << "," << this->y << ")" << std::endl;
  };
```

- What about initialization and destruction?
- Special member functions: constructors and destructors.

- Constructor: Same as class name (like, ClassName())
  - Things like “new” and initialization should go here
- Destructor: ~ClassName
  - Things like “delete” of memory should go here

```
Point( double ix=0., double iy=0.) { x=ix;y=iy;}
~Point(){}
```

- Then initialize

```
Point p1(0.,1.);
Point p2(2.,3.);
```

C++: Classes

- Members can be PUBLIC, PRIVATE, or PROTECTED:

- Public: Available to all classes
- Private: Available only to this class
- Protected: Available to derived classes (more later)

- Principle of least privilege: Make PRIVATE unless you need it publicly

- This is called the “public interface”
- The private bit is called the “implementation”
  - I like to append an underscore to the end of private implementation members

C++: Classes

- Example:

```
class Point {
 public:
  Point( double ix=0., double iy=0.) { x_=ix;y_=iy;}
  ~Point(){}

  void print() const {
    std::cout << "(" << x_ << "," << y_ << ")" << std::endl;
  };

  double x() const { return x_;}
  double y() const { return y_;}

private:
  double x_;
  double y_;

};
```

C++: Classes

- What about “const”?
- A constant object can be declared const
- Methods that MODIFY the class would not be…um… const.
- You need to tell the compiler which methods can be called on const objects:

```
void print() const {
    std::cout << "(" << x_ << "," << y_ << ")" << std::endl;
  };
```

C++: Operator Overloading

- Can REDEFINE operators for your type (“operator overloading”)
- For example, can define “+”, “-“, “+=“, and “-=“ to add or subtract two points

```
Point operator+( Point const & right ) const {
  Point retval( x_ + right.x_, y_ + right.y_ );
  return retval;
}
```

```
Point operator-( Point const & right ) const {
  Point retval( x_ - right.x_, y_ - right.y_ );
  return retval;
}
```

```
Point & operator+=( Point const & right )  {
  x_ += right.x_; y_ += right.y_ ;
  return *this;
}
```

```
Point & operator-=( Point const & right )  {
  x_ -= right.x_; y_ -= right.y_ ;
  return *this;
}
```

careful!
+ and - are const,
+= and -= are not const
return BY VALUE for + and -,
BY REFERENCE for += and -=

C++: Operator Overloading

- To use:

```
  Point sum = p1 + p2;
  Point dif = p1 - p2;
 sum += p1;
  dif -= p2;
```

C++: Operator Overloading

- Can overload all of these operators:
- Arithmetic: + - * / % += -= *= /= %=
- Bitwise logic: ^ & | ^= &= |= << >> >>= <<=
- Destructor: ~
- Assignment: =
- Logic : ! < > == != <= >= && ||
- Increment/decrement:++ --
- Dereferences: ->* ->
- Function calls: ( )
- Array indices: [ ]

- Will play with a few in your HW

C++: Classes and Scope

- Classes define a unique scope
- The functions of the classes are prepended with the scope.
- Example:
  - void Point::print() const

Header Files

C++: Definitions and Declarations

- Just like with functions, classes can have separate declarations and definitions
- Implementation (declarations) in header file
- Source (definitions) in a separate C++ file
- Then you can #include “Header.h”, and then LINK the objects together later.

Declare in header:

```
class Point {
 public:
  Point( double ix=0.,
         double iy=0.);
  ~Point();
  void print() const;
  double x() const;
  double y() const;
 private:
  double x_;
  double y_;
};
```

Define in separate file:

```
#include "Point.h"
Point::Point( double ix, double iy) {
  x_=ix;y_=iy;
}
Point::~Point(){}
void Point::print() const {
  std::cout << "(" << x_ << "," << y_ << ")" << std::endl;
};
double Point::x() const { return x_;}
double Point::y() const { return y_;}
```

C++: Header Files

- We've been using header files all along (#include <iostream>)

- In your homework you should make your own header file (StudentRecord.h) with the StudentRecord class in it.

- Then include into your "main" files with #include "StudentRecord.h"

- Note the "" versus <>:
  - "": Looks in current directory.
  - <>: Looks in default directories.

C++: Header Files

- Caveat! Can declare any number of times, so need to protect against multiple inclusion of code

- Use a preprocessor directive:

```
#ifndef Point_h
#define Point_h
class Point {
   (bla bla bla)
};
#endif
```

C++: Header Files

- A bit fancier:
  - DECLARE the class in the header file
  - DEFINE the class in the source file
  - COMPILE the source into an object library
  - LINK the “main” source file to the object library
  - RUN!

Hands on

- Go to “ClassExamples”:

  g++ -o read_points_example Point.cc read_points_example.cc -I.

  g++ -o read_points_example_strstream Point.cc read_points_example_strstream.cc -I.

- Or (better!) put it in a Makefile!

Makefiles

Makefiles

- Series of rules to execute in order:

Dependencies

Target

```
read_points_example: Point.cc read_points_example.cc
  g++ -o read_points_example Point.cc read_points_example.cc -I.
```

Rule

```
read_points_example_strstream: Point.cc read_points_example.cc
  g++ -o read_points_example_strstream Point.cc
read_points_example_strstream.cc -I.

all: read_points_example_strstream read_points_example

clean:
  rm *.o *~ read_points_example_strstream read_points_example
```

Makefiles

- Can also do all sorts of fancy things with Makefiles
  - You're encouraged to read about them but are not really responsible for writing them
- Example: Compile all the cc files in a directory and make executables (from "BasicExamples"):

```
CXX = g++
CXXFLAGS = -std=c++11

all: $(patsubst %.cc, %.out, $(wildcard *.cc))

%.out: %.cc Makefile
	$(CXX) $(CXXFLAGS) $< -o $(@:.out=)

clean: $(patsubst %.cc, %.clean, $(wildcard *.cc))

%.clean:
	rm -f $(@:.clean=)
```

Use g++

Use the C++11 standard

To make "all", compile cc files to ".out" exe files

Make the .out files, but don't use the ".out" suffix

Remove transients

Define "make clean" to use the "clean" statement