12.010 Computational Methods of Scientific Programming

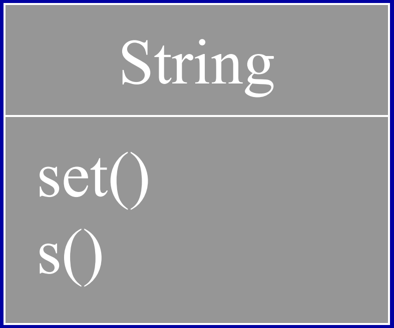
Lecturers
Thomas A Herring
Chris Hill

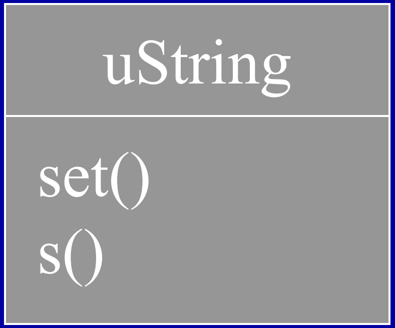
Summary

- Finished up C with structures and memory management
- Started with C++
  - C++ is C with the addition of “classes”
  - Class is a formal way to think about good program design.
    - Modularity, encapsulation, hierarchy, abstraction
  - A class has
    - Methods (program logic)
    - Data (variables)
    - can be private or public
- Today:
  - Example class in an example
  - Inheritance
  - Overloading (allows re-definition of methods for certain classes)

Inheritance

- Want new class uString. Like String except that the strings will be converted and stored in upper case. e.g.



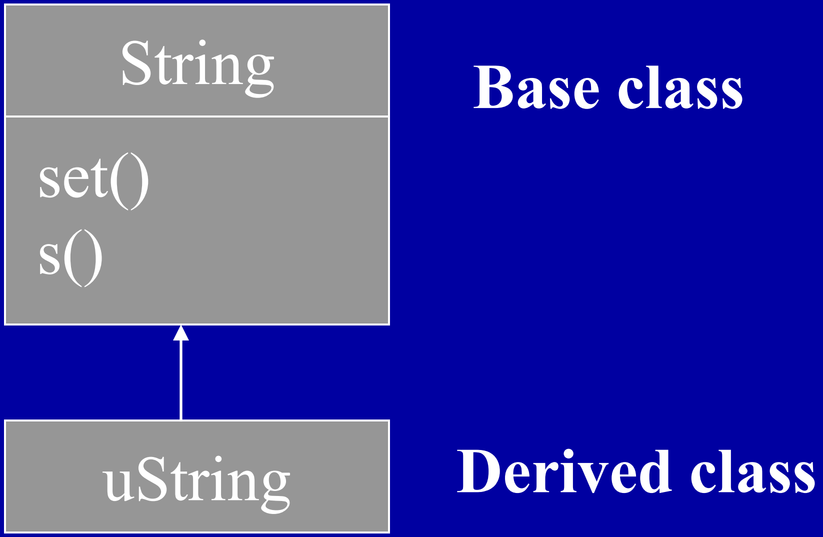


```
String s;
s.set("Hello");
printf("%s\n",s.s());
→Hello
```

```
uString s;
s.set("Hello");
printf("%s\n",s.s());
→HELLO
```

uString extends String

- No need to write uString from scratch.
- Inherit most code from String.
- Extend String::set to capitalise.
- A uString is a String with some extra feature.



C++ Inheritance Example

- New interface for uString

```
/* Extend String class to uString          */
/* uString stores strings as upper case */
class uString : public String {
 public:
  void set( char *);    /* Set a uString */
};
```

uString *set* method

```
/* Set str to point to a private copy of s */
void uString::set(char *s) {
 int i;
 String::set(s);
 for (i=0;i<strlen(s);++i) {
  if ( str[i] >= 'a' && str[i] <= 'z' ) {
   str[i] = toupper(str[i]);
  }
 }
}
```

Base class method

"protected" (not "private")

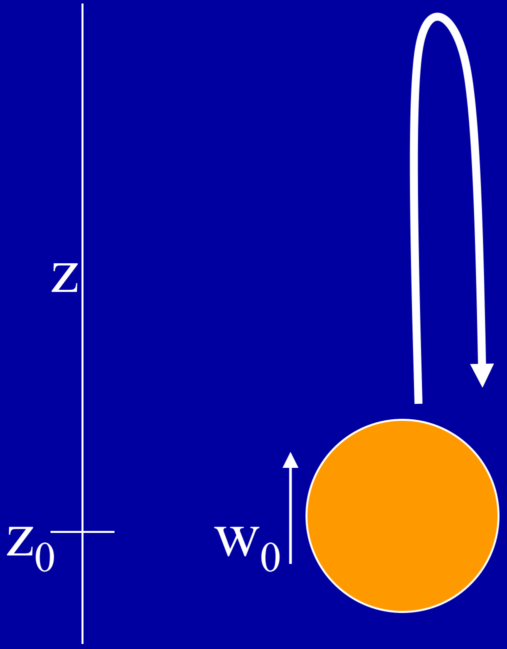
Return to concept of protected and private later in class

uString in action!

```
main()
{
 String  s1;
 uString s2;

 printf("Executable code starting\n");

 s1.set("Hello");
 printf("%s\n",s1.s());
 s2.set("Hello");
 printf("%s\n",s2.s());

 printf("Executable code ending\n");
}
```

Application Example

Throwing a ball in the air



$$w = w_0 + gt$$

$$z = z_0 + \int_0^t w\,dt$$

Get initial velocity and length of "experiment".

Calculate time evolution of w and z.

Print out "trajectory"

C “Procedural” Form

```
main (  )
{ float t=10.;  float w0=10.;
 t_gball *theBall;/* Stats for the ball          */

 /* Allocate space for full ball time history */
 createBall(w0, &theBall );
 /* Step forward the ball state */
 stepForwardState( t, &theBall );
 /* Write table of output */
 printTrajectory( t, w0, theBall);
}
```

C++ Using “Ball” Class

```
main()
{float w0 = 10.; float t=10.;
  Ball b;
  b.initialize(w0);
  b.simulate(t);
  b.printTrajectory();
}
```

All info. is held in “b”. Fewer args, cleaner “abstraction”.

C “Procedural” Form

```
main (  )
{ float t=10.;  float w0=10.;
 t_gball *theBall;/* Stats for the ball           */

 /* Allocate space for full ball time history */
 createBall(w0, &theBall );
 /* Step forward the ball state */
 stepForwardState( t, &theBall );
 /* Write table of output */
 printTrajectory( t, w0, theBall);
}
```

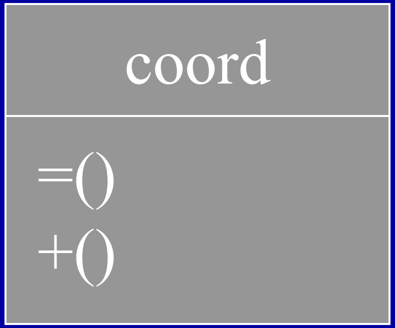
C++ Using “Ball” Class

```
main()
{float w0 = 10.; float t=10.;
  Ball b;
  b.initialize(w0);
  b.simulate(t);
  b.printTrajectory();
}
```

All info. is held in “b”. Fewer args, cleaner “abstraction”.

Overloading

Can redefine operators e.g. + to operate on classes e.g.



```
coord p1, p2, p3;
p3 = p1 + p2
```

This would then do

➔if p1=p2=(1,1,1) p3 = (2,2,2)

Overloading

- Have to define the meaning of + and = for a coord class object. Language defines meaning for integer, float, double etc but now we can define extra meanings.

```
class coord{
 public:
   coord operator+(coord );
private:
   int cx;   int cy; int cz;
};
```

```
coord coord::operator+ (coord c2)
{ coord temp;
  temp.cx = cx + c2.cx;
  temp.cy = cy + c2.cy;
  temp.cz = cz + c2.cz;
  return(temp);
}
```

Using namespace

- A namespace is a way of keeping classes and variables separate.
- The most common form seen in codes is:
  using namespace std
  which gives access to methods such as cout and cin
- Example:

  cout << “Message to screen” << endl
  will output the message and a new-line character. If the namespace is not used then
  std::cout << “Message to screen” << std::endl;
  will do the same thing

Example with variables

- Variables with the same name can have different values when used in different name space.
- Access to methods and variables is through syntax namespace::variables or namespace::method (later form is in cout example)
- Example of namespaces is in on code example link under instructions in the class web page
- Example code: http://geoweb.mit.edu/~tah/12.010/namespace.cc

Protected versus private

- These two declarations determine which methods can access a member of class.
- An example is on the “link to code examples” page given in the instructions part of the web page. http://geoweb.mit.edu/~tah/12.010/protect.cc
- Protected and private set which methods can change the values of variables.
- A protected method can not be directly access in a class but through inheritance and by being public the method can be accessed. If it is declared private an inherited class can not access it (replace protected with private and try to compile).

Conclusion

- C and C++: Characteristics similar to Fortran: Core program languages which are very powerful but programmer needs to do much of the work
  - Libraries of routines can be made and are available but these need to be carefully designed in C and Fortran (potentially routines can cause problems)
  - C++ classes minimize some of these problems but do not eliminate them completely.
  - Good modular program design can minimize problems.
- Remainder of class: Examine C++ examples and contrast Fortran and C if time available (see link on class web page) C_Basics.html C_Fortan_compare.html C_Pointers.html
- Homework 3 has been posted to web site (Due November 3, 2011)