CS319: Scientific Computing (with C++)

More on inheritance; Wrap-up

Week 12: **9am** and **4pm**, 29 March 2017

const

There are many aspects of the C++ language that we have not covered in this short course: there is an expectation that you will read beyond these notes.

But before we finish we'll study *interitance* in a little more detail. But first we'll consider the topic of **constants**.

A expression is constant it is value never changes. We are familiar with **literal consants**:

C++ also has three **keyword literals**: `true`, `false` and `nullptr`.

const

We can also define our own constants, by putting the keywork `const` before a "variable" declaration:

01const.cpp

```
  const double Pi=3.14159;
16 std::cout << "Pi=" << Pi << std::endl;

18 // Now change Pi
   Pi = 3.14; // This will not (and should not)  work
20
   std::cout << "Now Pi=" << Pi << std::endl;
```

Now, if we compile:

```
01const.cpp: In function ‘int main()’:
01const.cpp:20:6: error: assignment of read-only variable ‘Pi’
   Pi = 3.14;
      ^
```

We can add the `const` modifier to any data type, including a `class` that we define ourselves.

However, that creates issues... consider the following code, adpated from `Week11/02StaticVarInClass.cpp`

02constFunction.cpp

```
12  class Student
    {
14  private:
      std::string name;
16    static std::string university;
    public:
18    Student(std::string n) {name=n;};
      void set_name(std::string n) {name=n;};
20    void set_university(std::string u) {university=u;};
      std::string get_name(void) const {return(name);};
22    std::string get_university(void) const {return(university);};
    };
```

When writing a function header, we can designate some of the parameters as `const` meaning that the **copy** of the parameter that is passed cannot be changed in the function.

There are some times when this is necessary. Consider the following example...

03constParam.cpp

```
class Number
{
private:
  double x;
public:
  Number() {x=0.0;};
  Number(const Number &copy) {x=copy.x;}; // notice const
  void set_x(double X) {x=X;};
  double get_x(void) {return(x);};
  Number operator+(Number &b)
    {
      Number sum;
      sum.x = x + b.x;
      return(sum);
    }
};
```

03constParam.cpp

```
28  int main(void)
    {
30    Number a;
      a.set_x(121.234);
32    Number b(a+a);
      std::cout << "a.x= \t" << a.get_x() <<  std::endl;
34    std::cout << "b.x= \t" << b.get_x() <<  std::endl;
      return(0);
36  }
```

Why this works...

Recall... Inheritance

Last week, we introduced the concept of **inheritance**.

Inheritance

Given a class, construct from it a new class that has the properties of the original class, plus some new ones.

- The original class is called the **base**, **parent** or **super** class.
- The new one, is called the **derived**, **child** or **sub** class.

The derived class will inherit

- All the data members of the base class, though it may not be able to access them directly,
- The ordinary function members, but not constructors, destructors, or assignment operators.

Recall... Inheritance

Some programming/syntax aspects:

- The syntax for defining a derived class

```
class Derived : public Base
```

  Here `public` means that elements that were public to the base class are publicly accessible to the derived class. (This is the most typical case.)

```
class Derived : private Base
```

  Here `private` means everything remains private to the base class, and so can't be access by the derived class. (Not very useful).

When we first studied classes in C++, the only **access specifiers** we had were `private` and `public`.

Then we added **protected:**. These members act the same as private ones, except that they are directly accessible within a **derived** class.

Replacing members

Suppose, for example, that a base class has a method `void PrintData(void)`. It is possible to make a derived class that also contains a (new) method call `void PrintData(void)`.

The version of `PrintData(void)` from the derived class will superceed the one from the base class.

However, the original one does not disappear, it can still be called, but you need to use the base class name and scope resolution operator.

Note: If the base and derived classes have functions with the same names, but different **"signatures"**, then the function is overloaded, not replaced.

Replacing members

04Replacement.cpp

```
18  class Point {
    protected:  // this was private in 01InheritExample
20    int x;
    public:
22    Point(int X=0) {x=X;};
      int GetX(void) {return(x);};
24    void SetX(int X) {x=X;};
      void PrintData(void);
26  };

28  // The Point class has a method called "PrintData":
    void Point::PrintData(void)
30  {
      std::cout << "(" << x << ")\n";
32  }
```

Replacing members

04Replacement.cpp

```
36  class TwoPoint : public Point {
    private:
38    int y;
    public:
40    TwoPoint(int X=0, int Y=0) {x=X; y=Y;};
      int GetY(void) {return(y);};
42    void SetY(int Y) {y=Y;};
      void PrintData(void);
44  };

46  // The TwoPoint class also has a method
    // called PrintData()
48  void TwoPoint::PrintData(void)
    {
50    std::cout << "(" << x << "," << y << ")\n";
    }
```

Replacing members

04Replacement.cpp

```
54  int main(void)
    {
      TwoPoint d;
56
      d.SetX(1); d.SetY(2);
58    std::cout << "Calling the TwoPoint version of d.PrintData(): d=";
      d.PrintData();
60
      std::cout << "Calling the Point version of d.PrintData(): d=";
62    d.Point::PrintData();
      return(0);
64  }
```

There are other times when we need to take care which of two functions that have the same *signature* and belonging the base and derived class is called, and need to declare one of them as `virtual`, but that is beyond the scope of this class...

Your lab assignments, including the project will contribute 40% to your final grade for CS319.

The final exam contributes the other 60%.

The topics we have covered (not necessarily in order) are:

(a) Basic I/O (`cin`, `cout`), and manipulators (`endl`, `setw`, ...);

(b) Flow of control and looping (`if`, `for`, `while`, etc.)

(c) Fundamental data-types (`int`, `float`, `double`, `char`, `bool`, ...). Arrays.

(d) `string`, and C-style strings.

(e) Computer representation of numbers (underflow, overflow, machine epsilon, ...)

(f) Bit-wise operators

(g) Dynamic memory allocation, including of multidimensional arrays

(h) Functions: default parameter values, overloading, functions as arguments to other functions, recursion, pass by reference/value.

(i) Classes, including the `private`, `public` and `friend` access specifiers.

(j) Classes: constructors and destructors, including copy constructors.

(k) `template`s.

(l) Function and operator overloading (syntax, precedence, implicit/explicit arguments, the `this` pointer, unary/binary operators, assignment operators, ...).

(m) The C++ preprocessor

(n) The Standard Template Library (STL), including containers (especially `set`, `multiset` and `vector`), iterators, algorithms (but not functors); range-based loops.

(o) Optimisation and bisection;

(p) Sorting algorithms, and their complexities;

(q) Stacks, and their applications;

(r) Solving linear systems by the Jacobi and Gauss-Seidel methods;

(s) Solving diagonal and triangular systems (back-substitution);

(t) Sparse matrix representation, especially triplet and CCS; Matrix-vector multiplication

(u) Representation of graphs as adjacency matrices;

(v) PageRank and the Power Method;

(w) Reading to and writing from files; Comma Separated Values (CSV) files;

(x) Static variables, and class variables.

(y) `const`

(z) Inheritance (but not `virtual`).

. . . . . .

`Tip from last year:` Most marks lost (in assignments and exam) for not *explaining* how code works.

The final assessment for the module is a two hour exam during the "summer" sitting. The emphasis is more on C++ than Scientific Computing.

The paper has **four** questions: you should answer **any** three for full marks. All questions carry the same number of marks. The number of marks for each sub-question is indicated on the paper.

The paper features an short appendix listing some important C++ functions and objects, and giving a brief description. For example, it includes:

`fstream` (File stream)	
`ifstream`	Input file stream class
`ofstream`	Output file stream class
`open (char* f)`	Open the file named in a C-string
`close()`	Close the file

It does *not* include very fundamental elements of C++, such as the syntax of `for`-loops and `if`-statements.

The questions also includes several snippets of C++ code.

It is possible to get full marks for a piece of code that includes minor mistakes that a compiler would report (e.g., missing semicolon, using `=` to check equality).

I hope you have enjoyed CS319, and have learned something: I have!

Thank you for your commitment, collaboration, interest and insight: it has been a pleasure to teach this course.