110061

N95-16476

1994 Workshop on The Role of Computers in LaRC R&D

Object Oriented Numerical Computing in C++

John Van Rosendale

Institute for Computer Applications in Science and Engineering
jvr@icase.edu

Synopsis

C++ is an efficient object-oriented language of rapidly growing popularity. It can be of real value in a wide range of disciplines, including numerical computing, where it seems to offer important advantages over most competing languages.

Object-oriented languages

What exactly is an object-oriented language? The most important defining characteristic is support for "polymorphic data types." Procedural languages, like Fortran and C, contain built-in types such as integers, reals, characters and so on. The integer type, for example, consists of the requisite bits of data, a set of associated operations, +, *, /, ..., and coercions to and from the other built-in types. One can build data structures of arbitrary complexity in Fortran, but these are not "first class" types, like integers.

For example, one can form a "sparse_matrix" from arrays of integers indexing into arrays of reals. But Fortran 77 does not let one declare several of these as

```
sparse_matrix A,B,C
```

and then perform operations such as:

```
A = B + C
```

Languages like Clu and Ada, supporting "abstract data types," let one do precisely this. One can, for example, in Ada define a "set_of_words" abstract data type. This would be a user defined type which might be useful in comparing documents. Once the type is defined, one can then declare several such sets

```
set_of_words A,B,C
```

One can also operate on them just as with the built in types

```
A := B .+. C
```

where .+. might be a user-defined union operation.

OO languages push this concept further, allowing one to define a "set_of_<type T>", where T can be any type in the language. This new type, a "set_of_<type T>", is "first class" in OO languages, one can use variables of that type exactly like those of the built-in types. To make this clear, types are called "classes" in the OO world, while values of those types (classes) are called "objects," though whehter these new terms do more to clarify or obfuscate is not clear.

To see how OO ideas might be used in numerical computing, it might, for example, be useful to define a class "mesh_cell" which would be the basic unit of an unstructured mesh. Mesh cells come in a number of varieties, which can be thought of as subtypes (subclasses) of the type (class) "mesh_cell", as shown in Figure .

All mesh cells share certain properties, volume, temperature, pressure etc. declared as part of class "mesh_cell." Cubes and tetrahedrons share these properties, but have their own unique properties as well. They have different numbers of faces and vertices for example.

The ability to allow useful computing on a set of related but not identical user-defined types is the defining characteristic of OO languages. In the above case, one can make an array of "mesh_cells", consisting of prisms, tetrahedrons, and cubes. One can access the volume of any element of this array, since all "mesh_cells" have volume. To access specialized properties, one may have to select on the particular subclass of each "mesh_cell".

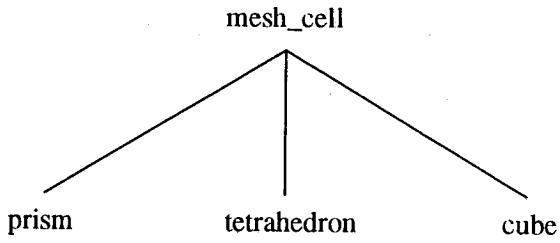


Figure 1: Mesh cell type hierarchy

C++ in numerical computing

How useful will C++ be in numerical computing? C++ contains most of the useful new features in Ada or Fortran 90, and is easily extensible in a number of ways. People around the world are rapidly developing class libraries for finite element analysis, for sparse matrix arithmetic, and so on. C++ together with a new class library is essentially a new application-specific language, and one that may have a powerful impact on a particular subdiscipline.

To see how this could have an impact, one need only realize that there are, for example, at least a dozen different unstructured grid codes here at Langley, with relatively little code shared between them. Given the appropriate class library supporting unstructured grids, one should be able to prototype new unstructured grid algorithms much faster, by borrowing large chunks of previously written code. This is the promise of OO computing in C++. Efficient execution, compatibility with previously written C and Fortran, and the OO approach are the major advantages to C++.

C++ also has its problems. One is that its syntax and semantics, inherited from C, are needlessly complex, significantly steeping the learning curve for new programmers. Another problem is that, like Ada and Fortran 90, C++ is a large language, full of complexities most programmers will never master. Only experts will master the full language, with most programmers limping along on their own particular subset.

These problems are real, but clearly not fatal, given the exponential growth of C++. From one perspective, C++ is essentially a halfway point between traditional procedural languages, like C and Fortran, and "rapid prototyping" languages like Smalltalk. Over the longer term, as computer power increases and our algorithms become more complex, one expects research numerical computing, like that done at Langley, to shift in the "rapid prototyping" direction. Use of C++ is an important step in that direction.

Object Oriented Numerical Computing in C++

1994 Workshop on The Role of Computers in LaRC R&D

John Van Rosendale

jvr@icase.edu

Institute for Computer Applications in Science and Engineering

An **object oriented** language is one allowing users to create a set of related types and then intermix and manipulate values of these related types.

- Such types are called **classes**.
- Values of such types are called **objects**.

Intermixing related user-defined types is called **polymorphism**. Polymorphism, and the things that go with it, *encapsulation*, *inheritance*, and *dynamic binding* give OO languages their increased semantic power.

Some Well Known OO Languages

Smalltalk

C++

Owl/Trellis

Eiffel

Objective C

Objective Pascal

Actors

Loops

Clos

Simula

Fortran is not OO. Integer variables, for example, have powerful properties the user cannot duplicate in new types. There is no way to define a sparse-matrix type, then do:

```
sparse_matrix A,B,C

A  =  B + C
```

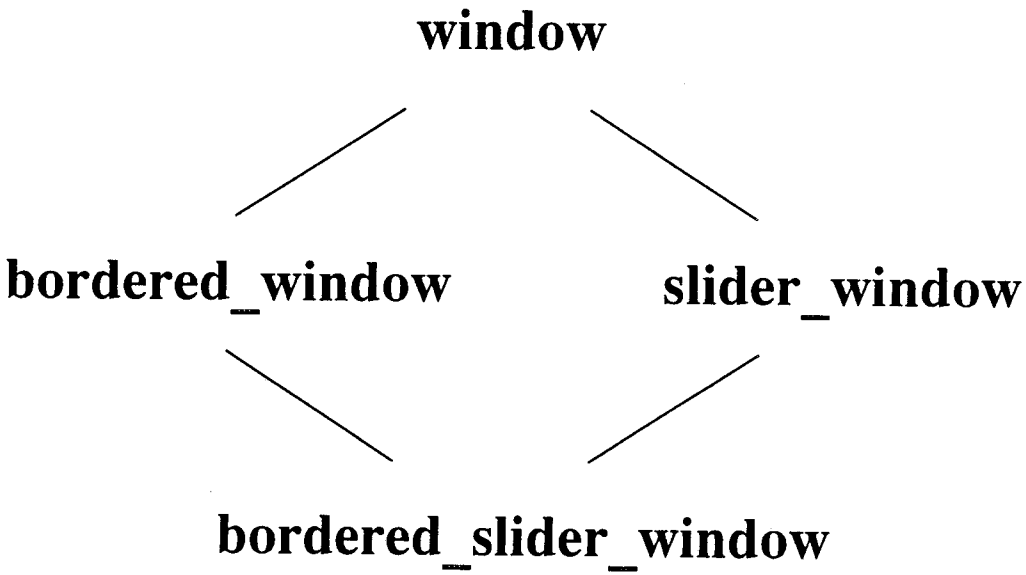
Languages supporting "abstract data types" allow users to define new types. In Ada, one can create a "sparse_matrix" type or a "set_of_words" type:

```
set_of_words A,B,C

A :=  B .+. C
```

where .+. is a user defined union operator.

Typical class lattice:



What is C++ ?

- A statically-checked object oriented language downward compatible with C
- Useful on any program with complex data structures (though originally intended for systems programming)
- Potentially useful in scientific programming, since it is efficient and supports complex data structures well

Templates – data abstractions with types as parameters

```
template<class T>
class stack {
   T* v;
   T* p;
   int sz;

   public:
       stack (int s){ y = p = new T[sz = s]; }
       ~stack () { delete[] v; }

       void push(T a){ *p++=a; }
       T pop(){ return *--p; }

       int size()  const { return p-v; }
};
```

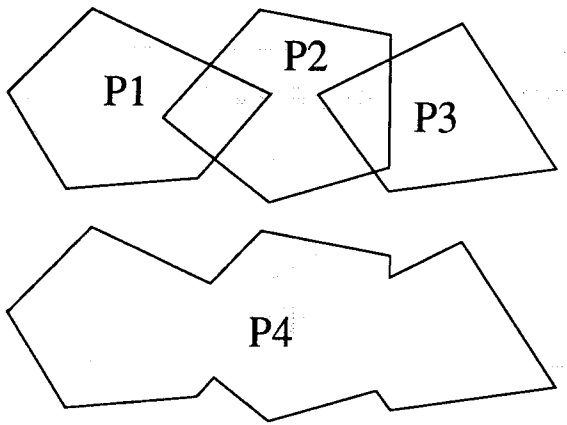
Operator Overloading – the ability to define new uses for the built-in operator symbols (+ - / ! = ...)

Example: polygon union

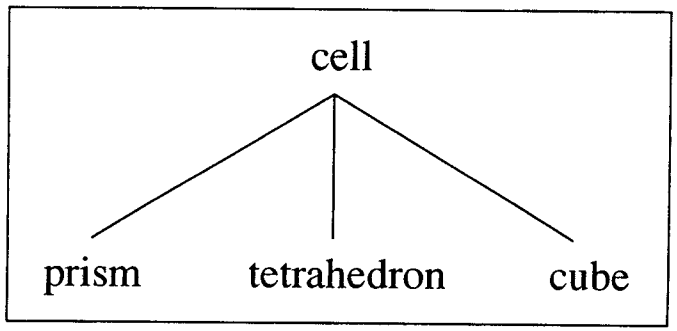
```
P4 = P1 + P2 + P3

P4 = union(union(P1,P2),P3)

P4 = P1.union(P2.union(P3))
```



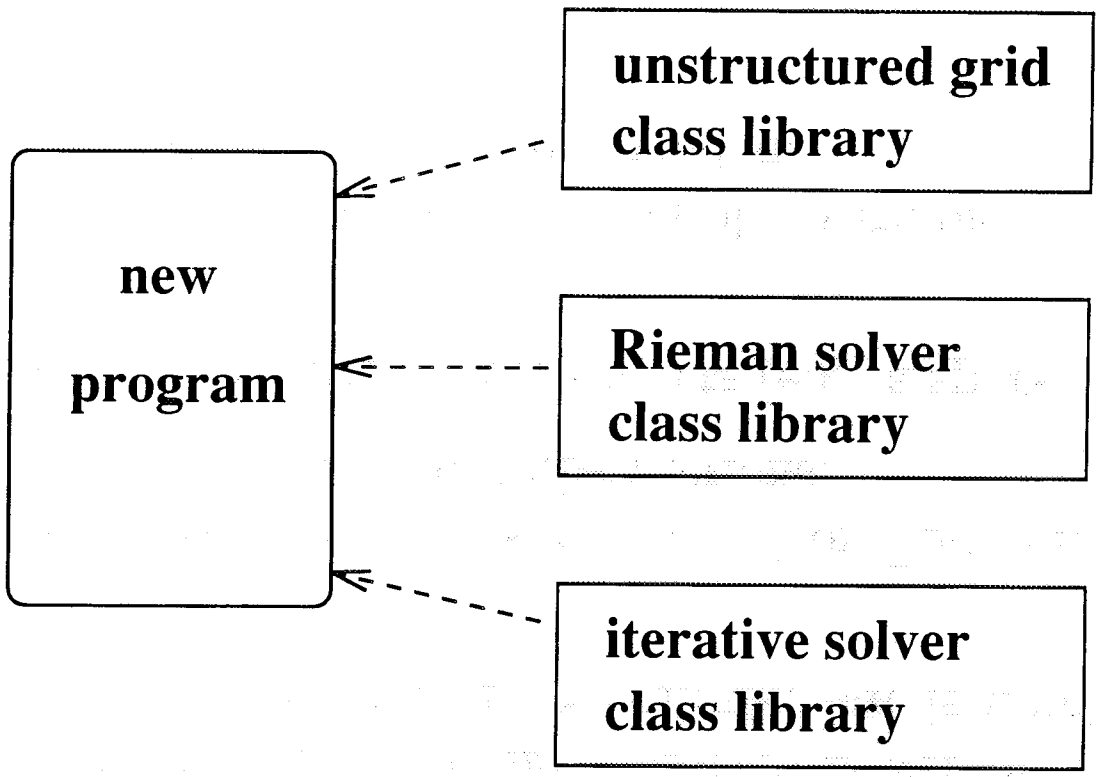
Using inheritance in numerical codes




```
cell:
        - volume
        - flow variables

cube:
        - 8 vertices
        - 6 faces
prism:
        - 6 vertices
        - 5 faces
```

Potential for code reuse:



Efficiency

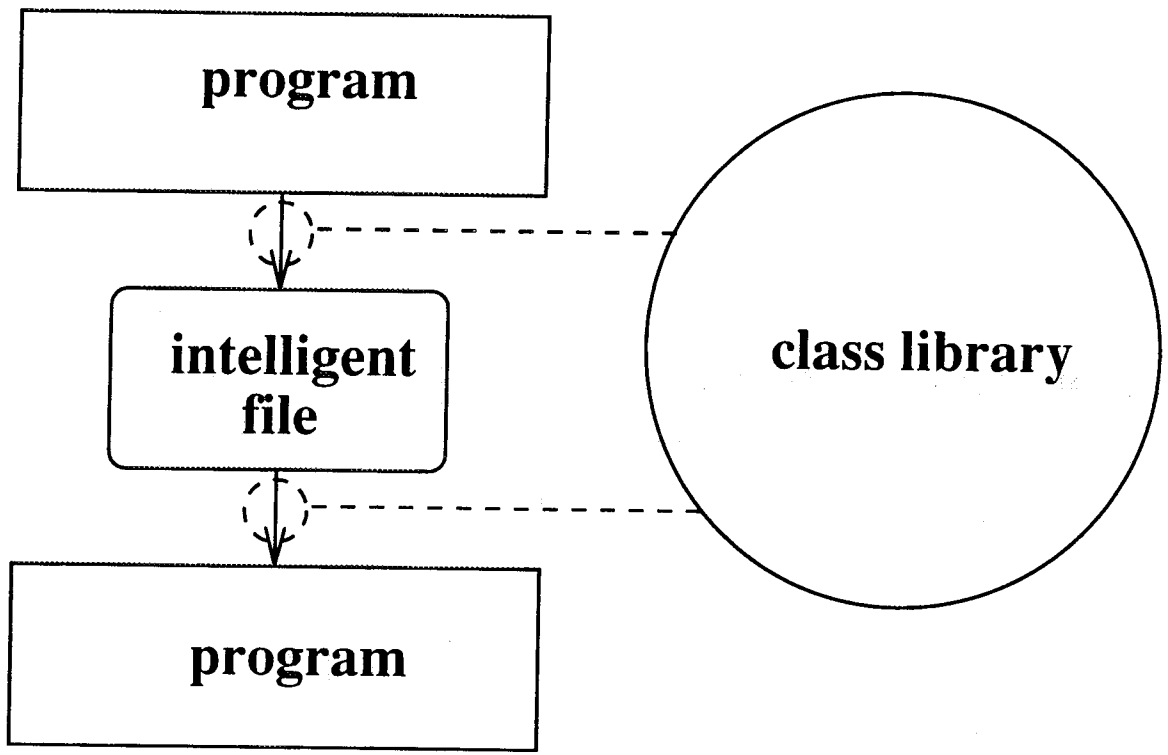
- C++ provides much of the semantic power of dynamically typed languages, like Smalltalk, but is much more efficient.
- C++ code runs about as fast as C or Fortran, if one avoids polymorphism and abstractions.
- The overhead in using large objects (e.g. a sparse matrix object) is minor.
- Fine grained objects, such as complex variables, are inefficient and should usually be avoided.

Observations

- Efficiency should be thought of in terms of the entire programming, debugging, and execution cycle.

- If a language would make programming substantially easier, there could be such a gain from algorithmic improvements, that significant run-time inefficiency could be tolerated.

Interoperability



Conclusions

- C++ is an effective OO language, and has become the defacto standard.
- C++ supports the data structures needed for complex numerical algorithms very well.
- It is efficient, and can be readily intermixed with C, Fortran and perhaps HPF.
- Developing numerical algorithms in C++ should increase opportunities for code reuse and for sharing code between programmers.