User-Defined Extractors and Inserters in C++

- By Cameron Hughes and Tracey Hughes

- May 7, 2004

Contents

The Stream Inserter Operator <<

Inserters are functions used to insert data or objects into an output source. The insertion operator for user-defined types must perform two levels of translations: It first breaks down the user-defined type into built-in datatypes, and then converts the built-in datatypes to a generic stream of bytes directed to some output device. The left bit Shift operator << is overloaded for the user-defined datatype, converting it to an output-translation operator. The function defines what the operation will do relative to the user-defined class. The function has access to the data members and functions of the class.

The appropriate representation of the class is then inserted into the output stream. In some cases, a class will have multiple representations and the insertion operator must be able to determine which one to use. This definition of the *inserter* should essentially maintain the semantics of the operator and should be consistent with expected syntax normally associated with the operator. The inserter is declared as a `friend` member function. An operator declared as a `friend` of a class has the advantage of having access to the private elements without being a member of the class. The definition of the `friend` function should take this general form:

```
ostream& operator<<(ostream& Out,const class_type &class_object)
{
  Out << class_object.data_member ... ; //insert object contents
  ...
  return Out;
}
```

To create an inserter for a user-defined object, this prototype is listed as part of the declaration for the class:

```
friend ostream& operator<<(ostream& Out, const class_type
&class_object);
```

The first parameter in the parameter list is a reference to an ostream. This is a reference because the stream will be altered once the representation of the `class_object` has been inserted into the stream. The ostream appears on the left side of the inserter operator. The second parameter in the list is the `class_object` that appears on the right side of the inserter operator. This is the user-defined object that's inserted into the stream. It returns a reference to the ostream to allow numerous inserters to be strung along into one message. Because the operator is a `friend` operator, the keyword precedes the prototype declaration.

NOTE

If the object is very large, use a reference to the object for the sake of efficiency. The object should be a `const` if the operation doesn't modify the object.

This is the prototype for our inserter of the user-defined class `course`:

```
friend ostream& operator<<(ostream& Out, course &Course);
```

Listing 3 shows the definition for the `friend` operator << for the `course` class:

Listing 3 Defining the `friend` Operator << for a User-Defined Object

```
ostream &operator<<(ostream &Out, course &Course)
{
  Out << Course.toHornClause() << ends;
  return Out;
}
```

For our `course` object, one of the appropriate representations is a horn clause. The member function `toHornClause()` is called. This member function actually creates the horn clause (not shown). The `toHornClause()` method returns a `string` that represents the `course` object as a horn clause of this form:

```
course(start_time,end_time,course_description,[days]).
```

This is an example of a horn clause representation of our `course` object:

```
course(1000,1200,"Introduction to Computer Architecture",
[M,W,F]).
```

This is what will be inserted into the output stream. The ostream is returned by the operator so a chain of insertion can be performed, as shown earlier in Listing 2:

```
DegreePlan << Schedule[N] << endl;
```

Two other appropriate representations for our `course` object are HTML and XML. We overload the inserter operator << to provide this translation as well as providing the horn clause translation.
In Listing 4, the inserter inserts the HTML tags and `course` object data into the output stream:

Listing 4 Defining an Inserter That Inserts HTML Tags and Object Data into an Output Stream

```
ostream &operator<<(ostream &Out, course &Course)
{
  Out << "<html>"  << endl  <<
     "<body>"  << endl  <<
     "<table>" << endl  <<
     "<tr>"   << endl  <<
     "<td>Start Time</td>" << endl <<
     "<td>"   << Course.startTime().c_str()  << "</td></tr>"
     << endl <<
     "<tr>"   << endl  <<
     "<td>End Time</td>"  << endl <<
     "<td>"   << Course.endTime().c_str()   << "</td></tr>"
     << endl <<
     "<tr>"   << endl  <<
     "<td>Description</td>"<< endl <<
     "<td>"   << Course.description().c_str() << "</td></tr>"
     << endl <<
      ...
     "</table>" << endl <<
     "</body>" << endl <<
     "</html>" << endl;
  return Out;
}
```

In Listing 5, the inserter inserts the XML tags and `course` object data into the output stream:

Listing 5 Defining an Inserter That Inserts XML tags and Object Data into an Output Stream

```
ostream &operator<<(ostream &Out, course &Course)
{
  Out << "<?xml version = /"1.0/" standalone=/"yes/"?>" << endl
<<
     "<DOCUMENT>"      << endl <<
     "<COURSE_START_TIME>"  << Course.startTime().c_str()  <<
     "</COURSE_START_TIME>" << endl <<
     "<COURSE_END_TIME>"   << Course.endTime().c_str()   <<
     "</COURSE_END_TIME>"  << endl <<
```

```
     "<COURSE_DESCRIPTION>" << Course.description().c_str() <<
     "</COURSE_DESCRIPTION>" << endl <<
     ...
     "</DOCUMENT>"      << endl;
  return Out;
}
```