5/10/2008

Common C++ Errors

These error statements were generated by g++ (Version?) on an HP workstation (Model?).
Originally: http://www2.hawaii.edu/~pautler/How-to/c++-compiler-errors.html

Compiler errors mention line numbers in a file. If you are using an emacs editor, a quick way to access line N is to go to the top of the file (Esc plus <) and then jump down (Ctrl-u, then N-1, then Ctrl-n).

Compile errors

table.C:7: parse error before '<'

If table.h declares a template class, then you're probably including table.cc in your compile command by mistake. Alternatively, you might be defining a template method with an unnecessary <...> after the :: and before the parameter list.

course.cc:26: parse error before '>'

If course.cc declares or instantiates a nested template object
(e.g., Table >;), then make sure that there is a space between the two >s or the compiler will interpret it as either the insertion or binary shift operator >> !

table.C:65: parse error before '&'

This explanation actually applies to any special character in a parse error, not just &. (The character used appears to be the first special character following the return type of the template function that appears on the line mentioned by the error).

If you have template functions in table.h, and you force all instantiations at once in one file (say, instan.cc), then you might try moving the table.h include statement closer to the end of all the includes. This change should allow the types stored in Table collections to be loaded before the collection itself is.

---

stu-views.cc:18: warning: implicit declaration of function `int setw(...)'

You haven't included the library that setw is in. (The compiler assumes all undefined functions return int and take (...) as parameters, so don't take these as clues.)

---

table.C:41: no matching function for call to `TableIterator<...>::TableIterator<...> (const Table<...> *, int)'

If you are passing 'this', or any other const variable, as an argument to a function that doesn't declare that parameter as const, then you could get this error. In this particular case, the TableIterator constructor was written to expect a regular Table*, not a const Table*, and that was fixed just by changing its parameter list.

foo.h:25: no matching function for call to
`Table >::
Table > (int, Ascending, bool (*)(const int *, const int *))'
table.C:10: candidates are:
Table >:: Table(int, Ascending, bool (*)(int *const &, int *const &))

You may be having trouble with your collection constructor because you declare it incorrectly. If your declaration here is:

Table<> > data;

You need to make it:

Table< color="#0000FF">const int*> > data;

---

collect2: ld returned 1 exit status
/bin/ld: Unsatisfied symbols:
DynArray::DynArray(int, int const &)(code)

Unsatisfied symbols mean that the linker can't find the definition of these operations. Possible reasons:

1. the .o containing them wasn't in the list of files you compiled,
2. the class they belong to is a template and wasn't instantiated (for the usual g++ reasons),
3. the operations are defined as inline in some source file rather than in a header file,
4. the method simply wasn't defined (don't trust the header file prototype).
   check that the method name is preceded by MyClass::
5. Tip: Make the method private and see who is trying to use it. According to Alex, "G++ has some problems with how templates mix with defaults, so it might be that you have to define some methods for classes for which you might not think you have to." For example, if class Barney contains a collection, its default op== might go undefined and you'll have to define it

yourself.

---

charts.cc: In method `class ChartData * ScoresChart::generateData()':
charts.cc:84: cannot lookup method in incomplete type `Student'

You've put a forward declaration for Student somewhere but not #included "student.h" in the .cc file where Student methods are used. In this particular case, "charts.h" does not have a forward declaration, but it does include "course.h" which has such a forward, so "charts.cc" must still do the include.
Alternatively, you might have put the forward declaration for Student inside Course, rather than outside and just before it, where it probably belongs.

---

foo.h:25: no matching function for call to
`Table::Table (int, bool (*)(const int *, const int *))'
table.C:10: candidates are: Table::Table(int, bool (*)(int *, int *))

The mismatch here is that the collection constructor expects const parameters to its boolean fn and it's not getting them. The fix is to change the template declaration of the collection -- Table. This assumes that the fn you're passing to the constructor already declares its parameters correctly.

---

course.cc: In function `static class Singleton* Singleton::Instance()':
course.cc:36: cannot call member function `Singleton::instanceExists() const' without object

If Instance() is a static member function, then 'this' is not available inside and you can't call other member functions of Singleton. You have to access data fields of the object directly.

---

stu-views.cc:140: bad argument 1 for function `Barney::Barney(int)' (type was class Barney)

You're missing a copy constructor for Barney.

charts.cc: In method `void Fred::findAndTally(class Label *)':
charts.cc:39: bad argument 2 for function Barney::FindIf(blah, func, ...) (type was {unknown type})

You might be passing FindIf a pointer to a template function, and you are hoping it will instantiate it with the appropriate type information. But the compiler is unable to infer how to instantiate the passed func. Add a forced-instantiation statement; alternatively, some compilers allow you to parameterize template fns when you call them.

---

charts.cc: In method `ScoreData::ScoreData()':
charts.cc:53: cannot allocate an object of type `ScoreLabel'
charts.cc:53: since the following virtual functions are abstract:
charts.cc:53: bool ChartLabel::operator ==(const class ChartLabel &) const

You probably defined subclass ScoreLabel's op== to take another ScoreLabel as its argument.

Unfortunately, the parameter types of virtualized fns must all match -- so change the type of the argument to ChartLabel, the base class. (Hopefully, ScoreLabel doesn't have any special data fields that need to be included in op== comparisons.)

---

g++ note on template function instantiation

If you've defined a template function, and there is one template parameter that only appears as the return type of the function, then g++ will fail to instantiate the function using that parameter even though you mention it as part of your explicit instantiation. One fix is to add a dummy argument to the function of the same type as the return value.

Run-time errors

Pid 10571 received a SIGSEGV for stack growth failure.

Possible causes: insufficient memory or swap space, or stack size exceeded maxssiz.

You're invoking an infinite regress of fn calls. If you're using the Singleton pattern, one way this can happen is by deleting the static member data inside a destructor; the proper way of freeing the static data is to write a static removeInstance() method that deletes the static data and which is called once, say at the end of main().

---

segmentation fault

There are many possible causes for this, but a subtle one occurs when you have container class B with overloaded ops such as +=. If one of your variables V is a B* and you apply += to it, be sure you're doing

((*V) += foo) and not (V += foo), otherwise pointer arithmetic will probably cause a segmentation fault.

Another likely cause is if you have any code like, elements = new T[len], where len could be 0, then you should replace it with this code,

```
if (len > 0)
    elements = new T[len];
else
    elements = 0;
```

because new[0] doesn't return 0.

Still another possible cause is that you seta string to 0 rather than "" as in, const char* name = 0, and then call a string method on it, for example,strlen(name).

In general, segmentation faults occur when you follow bad pointers, typically when something is deleted that wasn't allocated with new.

Floating exception (core dumped)

You probably tried to divide by zero.

Style tips

Don't use NULL in a C++ program. It's not officially defined as part of C++ and there is no reason to use it. An iterator is often a better choice than a pointer most places where you pass in or return a pointer (e.g., the result of a Find method).