Ticket #6631 (new Bugs)

BOOST_PP_ITERATION_FLAGS(): error: missing binary operator before token "("

Opened 3 years ago
Last modified 3 years ago

Reported by:	peter@…	Owned by:	no-maintainer
Milestone:	To Be Determined	Component:	preprocessor
Version:	Boost 1.49.0	Severity:	Regression
Keywords:	BOOST_PP_ITERATION_FLAGS	Cc:	blake-r@…, smr@…

Description

With Boost 1.49.0 and GCC, use of BOOST_PP_ITERATION_FLAGS() results in a compiler error:

```
# g++ -I $HOME/usr/rhel6-x86_64/boost_1_49_0/include -I. -o file -O2 file.cpp
In file included from file.cpp:1:0:
file.h:19:1: error: missing binary operator before token "("
file.h:24:1: error: missing binary operator before token "("
```

As a reproducable test case, I compiled the example given in the Boost.Preprocessor documentation:

```
// file.h
#if !BOOST_PP_IS_ITERATING

   #ifndef FILE_H_
   #define FILE_H_

   #include <boost/preprocessor/iteration/iterate.hpp>

   // 1st iteration:
   #define BOOST_PP_ITERATION_PARAMS_1 (4, (1, 10, "file.h", 0x0001))
   #include BOOST_PP_ITERATE()

   // 2nd iteration:
   #define BOOST_PP_ITERATION_PARAMS_1 (4, (1, 10, "file.h", 0x0002))
   #include BOOST_PP_ITERATE()

   #endif

#elif BOOST_PP_ITERATION_DEPTH() == 1 \
   && BOOST_PP_ITERATION_FLAGS() == 0x0001 \
   /**/


#elif BOOST_PP_ITERATION_DEPTH() == 1 \
   && BOOST_PP_ITERATION_FLAGS() == 0x0002 \
   /**/


#endif
```

```
// file.cpp
#include "file.h"
int main() {}
```

Both GCC 4.4 and GCC 4.6 fail to compile the example using Boost 1.49.0.

The above code compiles fine using Boost 1.48.0.

g++ (GCC) 4.4.6 20110731 (Red Hat 4.4.6-3)

g++ (GCC) 4.6.2 20110728 (prerelease)

▼ Attachments

Change History

Changed 3 years ago by Blake-R <blake-r@…> comment:1

- **Cc** *blake-r@…* added

Changed 3 years ago by patrick.hartling@… comment:2 follow-up: ↓ 3

Modifying `file.h` to be the following fixes the problem for me (testing with GCC 4.4 and Visual C++ 9.0 SP1):

```
// file.h
#if !BOOST_PP_IS_ITERATING

   #ifndef FILE_H_
   #define FILE_H_

   #include <boost/preprocessor/iteration/iterate.hpp>

   // 1st iteration:
   #define BOOST_PP_ITERATION_PARAMS_2 (4, (1, 10, "file.h", 0x0001))
   #include BOOST_PP_ITERATE()

   // 2nd iteration:
   #define BOOST_PP_ITERATION_PARAMS_1 (4, (1, 10, "file.h", 0x0002))
   #include BOOST_PP_ITERATE()

   #endif

#else
#if BOOST_PP_ITERATION_DEPTH() == 1 \
   && BOOST_PP_ITERATION_FLAGS() == 0x0001 \
   /**/

struct BOOST_PP_CAT(Y, BOOST_PP_ITERATION()) { };

#elif BOOST_PP_ITERATION_DEPTH() == 1 \
   && BOOST_PP_ITERATION_FLAGS() == 0x0002 \
   /**/

struct BOOST_PP_CAT(Z, BOOST_PP_ITERATION()) { };

#endif
#endif
```

The difference in structure is like this:

```
#if !BOOST_PP_IS_ITERATING
// ...
#else
#if BOOST_PP_ITERATION_DEPTH() == 1 \
   && BOOST_PP_ITERATION_FLAGS() == 0x0001 \
   /**/
// ...
#endif
#endif
```

Changed 3 years ago by Blake-R <blake-r@…> comment:3 in reply to: ↑ 2

Yes, it helps. But me interesting why?

Replying to patrick.hartling@…:

> Modifying `file.h` to be the following fixes the problem for me (testing with GCC 4.4 and Visual C++ 9.0 SP1):

```
// file.h
#if !BOOST_PP_IS_ITERATING

   #ifndef FILE_H_
   #define FILE_H_

   #include <boost/preprocessor/iteration/iterate.hpp>

   // 1st iteration:
   #define BOOST_PP_ITERATION_PARAMS_2 (4, (1, 10, "file.h", 0x0001))
   #include BOOST_PP_ITERATE()

   // 2nd iteration:
   #define BOOST_PP_ITERATION_PARAMS_1 (4, (1, 10, "file.h", 0x0002))
   #include BOOST_PP_ITERATE()

   #endif

#else
#if BOOST_PP_ITERATION_DEPTH() == 1 \
   && BOOST_PP_ITERATION_FLAGS() == 0x0001 \
   /**/

struct BOOST_PP_CAT(Y, BOOST_PP_ITERATION()) { };

#elif BOOST_PP_ITERATION_DEPTH() == 1 \
   && BOOST_PP_ITERATION_FLAGS() == 0x0002 \
   /**/

struct BOOST_PP_CAT(Z, BOOST_PP_ITERATION()) { };

#endif
#endif
```

The difference in structure is like this:

```
#if !BOOST_PP_IS_ITERATING
// ...
#else
#if BOOST_PP_ITERATION_DEPTH() == 1 \
   && BOOST_PP_ITERATION_FLAGS() == 0x0001 \
   /**/
// ...
#endif
#endif
```

Changed 3 years ago by Steve Robbins <smr@…> comment:4

- **Cc** *smr@…* added

I am currently investigating building all Debian packages with Boost 1.49 and "luabind" fails to build with very similar symptoms. Can the boost maintainer comment whether this is a bug in boost or a simple API break.

Thanks, -Steve

Changed 3 years ago by peter@… comment:5

This is the follow-up on the mailing list:

http://thread.gmane.org/gmane.comp.lib.boost.devel/228802