What does the compiler mean?
PDXCPP 2017-08-22

Unexpected EOF

```
struct S
{
    static void f();
}
```

```
test.cpp(5): fatal error C1004: unexpected end-of-file found
```

Long long long is too long

```
void f()
{
    long long long x;
}
```

```
error: ‘long long long’ is too long for GCC
     long long long x;
               ^~~~
```

Passing ‘x’ discards qualifiers

```
struct Container
{
    bool empty();
};


bool empty(auto &c)
{
    return c.empty();
}
```

```
error: passing ‘const Container’ as ‘this’ argument discards
qualifiers [-fpermissive]
     return c.empty();
                    ^
note:   in call to ‘bool Container::empty()’
     bool empty();
          ^~~~~
```

Passing ‘x’ discards qualifiers

- A non-static member method is also cv-qualified and that applies to the object pointed by `this`
- Can’t call a non-const method in a const object
- The template parameter was const

```
test.cpp: In instantiation of ‘bool
empty(auto:1&) [with auto:1 = const Container]’:

error: passing ‘const Container’ as ‘this’
argument discards qualifiers [-fpermissive]
```

Need typename

```
template <typename T> struct S
{
    T::Type t;
};
```

```
error: need ‘typename’ before ‘T::Type’ because ‘T’ is a
dependent scope
     T::Type t;
     ^
```

Need typename

- In order to print the warning, the compiler needs to determine that it would be correct with `typename`, so why complain?
- C++ Standard says it’s necessary to disambiguate dependent types from non-types
- P0634R0 (“Down with `typename`!”) proposes to remove the need where only a type is possible

Member was not declared

```
template <typename T> struct Base
{
    int i;
};

template <typename T> struct S : Base<T>
{
    void g() { i = 0; }
};
```

```
Clang
error: use of undeclared identifier 'i'

GCC
error: ‘i’ was not declared in this scope
     void g() { i = 0; }
                ^
```

Declaration must be available

```cpp
template <typename T> struct Base
{
    void f();
};

template <typename T> struct S : Base<T>
{
    void g() { f(); }
};
```

```
Clang
error: use of undeclared identifier 'f'

GCC
error: there are no arguments to ‘f’ that depend on a template
parameter, so a declaration of ‘f’ must be available [-
fpermissive]
     void g() { f(); }
                ^
```

Declaration must be available

- S<T>’s base depends on the template argument T
  - Accesses to the base must happen at second phase lookup (Argument Dependent Lookup)
  - Therefore, accesses to base must either “depend on a template parameter” or be qualified:

```
this->i = 0;
this->f();
Base<T>::i = 0;
Base<T>::f();
```

Weird error 1

```
struct S
{
    static void f();
};

int main()
{
    S;:f();
}
```

```
error: declaration does not declare anything [-fpermissive]
     S;:f();
     ^
error: expected primary-expression before ‘:’ token
     S;:f();
       ^
```

Weird error 2

```
bool unix = false;
bool windows = false;
void sysident()
{
#ifdef _WIN32
    windows = true;
#else
    unix = true;
#endif
}
```

```
error: expected unqualified-id before numeric constant
 bool unix = false;
      ^
In function ‘void sysident()’:
error: lvalue required as left operand of assignment
     unix = true;
            ^~~~
```

Weird error 2

- When the error makes no sense, try reading the preprocessor output

```
$ gcc -E /tmp/test.cpp
# 1 "/tmp/test.cpp"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "/tmp/test.cpp"
bool 1 = false;
bool windows = false;
void sysident()
{



    1 = true;

}
```

Unable to find string literal operator

```
#define MAKE_HTML(x) "<html><body>"x"</html></body>"
const char *text()
{
    return
MAKE_HTML("<p>List:</p><ul><li>One</li><li>Two</li></ul>");    // error!
}

const char *text2()
{
    return "<html><body>"
           "<p>List:</p><ul><li>One</li><li>Two</li></ul>"
           "</html></body>";
}
```

```
error: unable to find string literal operator ‘operator""x’ with
‘const char [27]’, ‘long unsigned int’ arguments
 #define MAKE_HTML(x) "<html><body>"x"</html></body>"
                                    ^

note: in expansion of macro ‘MAKE_HTML’
     return MAKE_HTML("return
MAKE_HTML("<p>List:</p><ul><li>One</li><li>Two</li></ul>");    // error!
            ^~~~~~~~~
```

Unable to find string literal operator

- String literal concatenation usually requires no space:

```
static const char text[] = u8”a””b”;
```

- Before C++11, it worked in macros too
- C++11 introduced User Defined Literals, so the parsing changed
- Clang error message (with -std=c++11):

```
error: invalid suffix on literal; C++11 requires a space
between literal and identifier [-Wreserved-user-defined-
literal]
```

Invalid use of incomplete type

```
#include <QTcpSocket>

void doConnect(const QString &addr)
{
    QTcpSocket *socket = new QTcpSocket;
    socket->connectToHost(addr);
    socket->waitForConnected();
    qDebug() << socket->peerAddress().toString() << "connected";
}
```

```
error: invalid use of incomplete type ‘class QHostAddress’
     qDebug() << socket->peerAddress().toString() << "connected";
                                     ^
```

Ambiguous overload with built-in

```
#include <string>
struct JsonNode {
    JsonNode();
    JsonNode(int);
    JsonNode(std::string const &);

    operator int();
    operator std::string const &();

    JsonNode &operator[](size_t);
    JsonNode &operator[](std::string const &);
};

void test4() {
    JsonNode v;
    v["abc"] = 123; // doesn't compile!
}
```

```
error: ambiguous overload for ‘operator[]’ (operand types are
‘JsonNode’ and ‘const char [4]’)
note: candidate: operator[](long int, const char*) <built-in>
note: candidate: JsonNode& JsonNode::operator[](const string&)
```

Ambiguous overload with built-in

- There are two viable conversions for

```
operator[](v, ”abc”)
```

1) Convert v to int:

```
operator[](v.operator int(), ”abc”)  →
”abc”[int(v)]
```

2) Convert "abc" to std::string:

```
operator[](v, std::string(”abc”))
```

MSVC can’t count

```
[same code]
```

```
  test.cpp(17): error C2666: 'JsonNode::operator []': 2 overloads have
  similar conversions
1 test.cpp(12): note: could be 'JsonNode &JsonNode::operator [](const
  std::string &)'
2 test.cpp(11): note: or       'JsonNode &JsonNode::operator [](::size_t)'
3 test.cpp(17): note: or       'built-in C++ operator[(__int64, const char
  [4])'
  test.cpp(17): note: while trying to match the argument list '(JsonNode,
  const char [4])'
  test.cpp(17): fatal error C1903: unable to recover from previous error(s);
  stopping compilation
  Internal Compiler Error in C:\Program Files (x86)\Microsoft Visual
  Studio\2017\BuildTools\VC\Tools\MSVC\14.10.25017\bin\HostX64\x64\cl.exe.
  You will be prompted to send an error report to Microsoft later.
  INTERNAL COMPILER ERROR in 'C:\Program Files (x86)\Microsoft Visual
  Studio\2017\BuildTools\VC\Tools\MSVC\14.10.25017\bin\HostX64\x64\cl.exe'
      Please choose the Technical Support command on the Visual C++
      Help menu, or open the Technical Support help file for more information
```

ISO C++ says that worst is better than the worst

```
void function(char x, double y);
void function(int x, int y);

int main() {
    function('a', 'b');
    return 0;
}
```

```
warning: ISO C++ says that these are ambiguous, even though the
worst conversion for the first is better than the worst
conversion for the second:
/tmp/test.cpp:2:6: note: candidate 1: void function(int, int)
/tmp/test.cpp:1:6: note: candidate 2: void function(char,
double)
```

ISO C++ says that worst is better than the worst

- It’s ambiguous because there’s no perfect match
  - Note GCC inverted the order of the overloads
  - For the first overload, both `’a’` and `’b’` need to be converted to `int`
  - For the second, `’a’` matches `char` perfectly, but conversion of `’b’` to `double` is really bad
  - The “worst conversions for the first” (from `char` to `int`) are “better than the worst conversion for the second” (from `char` to `double`)

C++ does not support default-int

```cpp
struct S
{
    static S f();
};

struct T : S
{
    int i;
}

S f()
{
    return S::f();
}
```

```
test.cpp(11): error C2146: syntax error: missing ';' before
identifier 'f'
test.cpp(12): error C4430: missing type specifier - int assumed.
Note: C++ does not support default-int
test.cpp(13): error C2440: 'return': cannot convert from 'S' to 'int'
test.cpp(13): note: No user-defined-conversion operator available
that can perform this conversion, or the operator cannot be called
test.cpp(14): error C2617: 'f': inconsistent return statement
test.cpp(11): note: see declaration of 'f'
```

(X+c) < X is always false

```
int addOne(int v1)
{
    int sum = v1 + 1;
    if (sum < v1)
        throw Overflow();
    return sum;
}
```

```
warning: assuming signed overflow does not occur when
assuming that (X + c) < X is always false [-Wstrict-
overflow]
```

(X+c) < X is always false

- C and C++ standards say signed integer overflow is Undefined Behaviour
- In a well-formed program, you don't ever overflow, so X + c must be >= X
- Solutions:
  1) Use unsigned (if it makes sense)
  2) Avoid the UB

```
if (v1 == std::numeric_limits<int>::max())
    throw Overflow();
return v1 + 1;
```

- GCC prints this warning due to a bug report long ago (deprecated in GCC 8)

Will break strict aliasing rules

```
uint g(float f)
{
    return *reinterpret_cast<uint *>(&f);
}
```

```
warning: dereferencing type-punned pointer will break
strict-aliasing rules [-Wstrict-aliasing]
```

Will break strict aliasing rules

- Same as the previous warning: GCC prints because of complaints in bug reports
- C and C++ Standards say that two distinct types (other than `char`s) cannot alias each other
- What does this return?

```
int f(int *iptr, short *sptr)
{
    *iptr = 42;
    *sptr = 0;
    return *iptr;
}
```

malloc argument out of range

```
#include <stdlib.h>

void *f(int n)
{
  return malloc(n > 0 ? 0 : n);
}
```

```
64-bit
warning: argument 1 range [18446744071562067968,
18446744073709551615] exceeds maximum object size
9223372036854775807 [-Walloc-size-larger-than=]
32-bit
warning: argument 1 range [2147483648, 4294967295] exceeds
maximum object size 2147483647 [-Walloc-size-larger-than=]
```

malloc argument out of range

- `malloc()` takes `size_t`
- `size_t` is defined in [support.types.layout]/3 as:

  “The type `size_t` is an implementation-defined unsigned integer type that is large enough to contain the size in bytes of any object.”

- But the maximum size is limited by `ptrdiff_t` to half that ([support.types.layout]/2):

  “The type `ptrdiff_t` is an implementation-defined signed integer type that can hold the difference of two subscripts in an array object, as described in 8.7.”